Global Optimisation in Process Design

Robert Paul Byrne

A thesis submitted for the degree of Doctor of Philosophy of the
University of London

Department of Chemical & Biochemical Engineering
University College London
London WC1E 7JE

August 1997

Abstract

This thesis concerns the development of rigorous global optimisation techniques and their application to process engineering problems. Many Process Engineering optimisation problems are nonlinear. Local optimisation approaches may not provide global solutions to these problems if they are nonconvex.

The global optimisation approach utilised in this work is based on interval branch and bound algorithms. The interval global optimisation approach is extended to take advantage of information about the structure of the problem and facilitate efficient solution of constrained NLPs using interval analysis. This is achieved by reformulating the interval lower bounding procedure as a convex programming problem which allows inclusion of convex constraints in the lower bounding problem. The approach is applied to a number of standard constrained test problems indicating that this algorithm retains the wide applicability of the interval methods while allowing efficient solution of constrained problems.

A new approach to the construction of modular flowsheets is developed. This approach allows construction of flowsheets from linked unit models which enable the application of a number of global optimisation algorithms. The modular flowsheets are constructed with 'generic' unit operations which provide interval bounds, linear bounds, derivatives and derivative bounds using extended numerical types. The genericity means that new 'extended types' can be devised and used without rewriting the unit operations models.

The new interval global optimisation algorithm is applied to the generic modular flowsheet. Using interval analysis and automatic differentiation as the arithmetic types, lower bounding linear programs are constructed and used in a branch and bound framework to globally optimise the modular flowsheet.

Contents

1 Introduction **8**

1.1 Global Optimisation 9

1.1.1 Problem Statement 9

1.1.2 Classification of Optimisation Methods 11

1.2 Process Flowsheeting 12

1.2.1 Equation Oriented Flowsheeting 12

1.2.2 Modular Flowsheeting 13

1.3 Thesis Overview 13

2 Nonrigorous Global Optimisation **15**

2.1 Random Search Optimisation 15

2.1.1 Pure Random Search 17

2.1.2 Augmented Local Search 17

2.1.3 Adaptive Random Search 18

2.2 Generalised Descent 21

2.2.1 Trajectory Methods 21

2.2.2 Penalty Methods 23

2.3 Summary 24

3 Rigorous Global Optimisation **26**

3.1 The Form of the Covering Algorithm 27

3.2 Relaxation Methods 28

3.2.1 Benders Decomposition 30

3.2.2 Convex Underestimation 37

3.3 Bound & Exclude Methods 45

3.3.1 Lipschitz Optimisation 46

3.3.2 Interval Methods 49

3.4 Summary 60

3.4.1 Efficiency and Applicability of Bounding Methods 61

3.4.2 Interval Analysis Methods 62

4 Improved Interval Optimisation **66**

4.1 Branch and Bound in Interval Methods 66

4.1.1 Partitioning a Box 67

4.1.2 Location of a Single Minimiser 69

4.1.3 A Depth-first Interval Algorithm 70

4.2 Symbolic Information and Structure 73

4.2.1 Reformulation of NCP 74

4.2.2 Relaxed Problems from Reformulation 77

4.2.3 Linear Relaxations of NCP 85
4.3 Algorithms 89
4.3.1 The IGOR Algorithm 89
4.3.2 The RIGOR Algorithm: Reduction 91
4.3.3 Results of Application to Test Problems 93
4.4 Summary 98

5 Optimisation of Modular Systems **100**
5.1 Motivating Example 101
5.2 Formulation of Modular Flowsheets 103
5.2.1 The General Unit Module 103
5.2.2 Different Formulations 105
5.3 Extended Type Flowsheets 106
5.3.1 Extended Arithmetic Types 107
5.3.2 Model Genericity 108
5.4 Interval Modular Flowsheets 109
5.4.1 Interval Unit Operations 109
5.4.2 Optimisation of the Interval Flowsheet 111
5.5 Optimisation of Extended Flowsheets 113
5.6 Solving the Recycle Problem 116
5.6.1 Sequentially 117
5.6.2 Simultaneously 118
5.7 Summary 119

6 Conclusion **120**

6.1 Global Optimisation Algorithms 120

6.1.1 Future Work 121

6.2 Optimisation of Modular Systems 123

6.2.1 Future Work 124

A Mathematical Definitions **133**

A.1 Relevant Properties of Sets and Functions 133

A.1.1 Continuity 133

A.1.2 Convexity 133

A.1.3 Convex Hull/Envelope 134

A.1.4 Lipschitz Condition 135

B Extended Arithmetic Types **136**

B.1 Automatic Differentiation 136

B.2 Linear Underestimator Arithmetic 138

C Selected Test Problems **140**

D Nomenclature **148**

List of Figures

2.1 The Random Search Methods 16

2.2 Augmenting Local Search 17

2.3 Adaptive Random Search 19

2.4 Generalised Descent Methods 21

2.5 Trajectory Methods 22

3.1 Rigorous Optimisation Methods 27

3.2 The Relaxation Methods 29

3.3 Linearised Lagrangian underestimators 35

3.4 GOP Relaxed Master Problem 36

3.5 Convex Underestimation Methods 37

3.6 The convex envelope of a separable function. 39

3.7 The convex envelope of a product term 42

3.8 The Bound & Exclude Methods 46

3.9 Lipschitz underestimators using Piyavskii's Algorithm. 48

3.10 Geometric interpretation of the Interval Newton Algorithm 55

4.1 Relaxations of $x^2 + 10$ 87

4.2 NE Relaxations of $x^2 + 10$ on $[-2, 0]$ and $[0, 3]$ 88

5.1 The Haverly Pooling Problem 101

5.2 A General Unit Module 104

5.3 A Tear Unit Module 105

5.4 A generic model 108

5.5 Black Box Models 109

5.6 Unit Operations Based on Interval Arithmetic 110

5.7 Optimisation of An Interval Flowsheet 111

5.8 Torn Recycle Stream in an Interval Flowsheet 116

C.1 Reactor Network Optimisation 142

C.2 Design of a Three Stage Process with Recycle 143

List of Abbreviations

AD Automatic Differentiation 136

EO Equation Oriented .....

GBD Generalised Benders Decomposition 30

GBM Generalised Benders Master problem 32

GBP Generalised Benders Primal problem 31

GOP The GOP algorithm 33

IRD Inner Relaxed Dual 35

MILP Mixed Integer Linear Program .....

MINLP Mixed Integer Nonlinear Program .....

MVNE Combination of MV and NE relaxations 94

MV Relaxation based on the Mean Value Inclusion 83

NCP Nonconvex Program 10

NE Relaxation based on the Natural Extension 81

NLP Nonlinear Program 10

SM Sequential Modular 100

Chapter 1

Introduction

Process design and operation are complex tasks for which it is not easy to find a 'best' solution. The application of optimisation at the design stage can have a significant impact on the profitability of a process at a comparatively low cost. This is helped by the maturity of nonlinear programming (NLP) techniques which have been an active research front in Process Engineering for many years. However, these techniques can only guarantee that global minima will be found if the process model is convex. If the model is nonconvex a local optimisation algorithm may not locate a global optimum and, even when it does, there is no guarantee that the solution is global. It can no longer be assumed that the design is the best possible design because the solution to the optimisation problem may not be the global solution.

In some cases the choice between applying local versus global optimisation is simple. If the model is known to be convex then local optimisation will provide global solutions and so convex/local optimisation should be applied. If the problem is nonconvex and the global solution absolutely must be located to solve the problem, for example in molecular conformation, Gibbs energy and certain integer problems, then global optimisation is the only choice. Process design problems do not generally fall into either of these groups, models describing complex processes can have nonconvexities which may or may not affect the solution so the confidence in solutions obtained by local methods is reduced, and a global approach may be appropriate.

On the other hand, processes which make a profit, do so independently of the rigour of solution – if the local optimisation locates the global solution then a global optimisation algorithm will only find the same one – so application of a local optimisation algorithm from a handful of starting points may be the best option.

If a global optimisation approach has reliability and efficiency close to that of the local optimisation then the rigorous global optimisation approach will certainly be the best. And that is the reason that this thesis considers the design of global optimisation algorithms and their application to Process Engineering problems.

1.1 Global Optimisation

The advantages of optimisation are well known, it is an attempt to provide the best possible solution to a problem. Thus the Design Engineer can be confident that the design produced is the best one for the problem. Traditionally, however, this has not been the case if the problem is not convex. The problem may exhibit more than one locally optimal solution.

Problems from many areas are nonconvex and possibly multiextremal including network planning problems, economy of scale problems, minimisation of Gibbs energy, molecular conformation problems, VLSI chip design and protein folding. Techniques for determining the minimiser of a convex optimisation problem are well established but procedures for solving nonconvex optimisation problems and determining global minima are not so well developed, widely used, or well documented.

1.1.1 Problem Statement

Where appropriate this thesis uses standard nomenclature for optimisation problems and algorithms.

The general nonconvex optimisation problem is stated in the same way as a convex problem. The difference is that there are no implicit assumptions about convexity,

continuity or differentiability.

The Nonconvex Program NCP is stated as follows

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & \\ g(x) &\leq 0 \\ h(x) &= 0. \end{aligned} \tag{NCP}$$

Where x is a n-vector of *independent variables*, $f : \mathbb{R}^n \to \mathbb{R}$ is called the *Objective Function* and

$$\mathcal{A} = \{x \in \mathbb{R}^n | h(x) = 0, g(x) \leq 0\} \tag{1.1}$$

is the *Feasible Region* defined by the equality constraints, $h : \mathbb{R}^n \to \mathbb{R}^{n_h}$, and inequality constraints, $g : \mathbb{R}^n \to \mathbb{R}^{n_g}$. This problem will be referred to as a Nonconvex Programming Problem (NCP).

A global minimisers, x^*, is defined by

$$f(x^*) \leq f(x) \qquad \forall x \in \mathcal{A} \tag{1.2}$$

For nonconvex problems x^* is not necessarily unique; there may be more than one global minimiser. However, in many cases a single member of this set is adequate as a solution. Given that every global minimiser is a local minimiser the necessary conditions for a local minimum are also necessary for a global minimum.

As computer floating point arithmetic does not produce 'exact' results ϵ-global minimisers are defined

$$f(x^*) - f(x) \leq \epsilon \qquad \forall x \in \mathcal{A} \tag{1.3}$$

where ϵ is a small constant and the equality constraints are relaxed by epsilon.

Bounds on Variables Rigorous global optimisation is based on obtaining bounds on parts of the problem when the independent variables lie within certain bounds. Throughout this thesis upper and lower bounds are represented by over and under bars respectively. Thus $\overline{x}$ is an upper bound on x, $\underline{\mathrm{y}}$ is a lower bound on y and $\overline{y^*}$ is an upper bound on y^*.

Intervals or ranges are denoted by capital letters such that an interval $X = [\underline{x}, \overline{x}]$ implies that $\underline{x} \leq x \leq \overline{x}$. When x is a vector X is a box, nonrectangular regions are denoted with a calligraphic typeface such as $\mathcal{X}$, in particular the feasible region is denoted by $\mathcal{A}$.

1.1.2 Classification of Optimisation Methods

It is useful, in general, to treat constrained and unconstrained optimisation separately. Whilst this is the most natural distinction to make for convex optimisation algorithms the guarantee of a global solution by a given method is a more useful primary classification to make in nonconvex optimisation. This is because methods which guarantee global optimality may require assumptions which are not strictly necessary in other methods and because it is unrealistic to compare the computational effort required by a global algorithm, which must acquire considerably more information about the problem, with one which cannot provide such a guarantee.

1.1.2.1 Nonrigorous Global Optimisation

These methods provide the possibility of solving global optimisation problems where local algorithms may fail, that is: no assumptions about the convexity of the objective are made in the formulation. This does not mean, however, that the global solution will definitely be found. The important advantage over the rigorous methods is that they can make fewer assumptions about the problem, allowing a wider range of problems to be attempted. The disadvantage is simple; the solution obtained cannot be known, or proven, to be the global minimum.

1.1.2.2 Rigorous Global Optimisation.

Once certain assumptions about the problem structure have been made, for example an upper limit on the growth of the objective function, it is possible to guarantee a global solution (or set of solutions) to the problem. The advantage of an assured global solution is clear but the assumptions which need to be made may not be valid for all problems. In some cases, notably interval techniques, the assumptions required can be relaxed at the expense of efficiency.

1.2 Global Optimisation in Process Flowsheeting

Computer simulation of chemical processes is an important and widely used tool in the operation of existing plants and the design of new ones. Optimisation of these flowsheets can have significant economic benefits for the operation of a process by identifying and implementing optimal operating conditions early in the design.

Two main approaches to process simulation and optimisation have emerged. Equation based flowsheets model the flowsheet as a large system of nonlinear equations, derived from mass and energy balances, representing the whole process. Modular simulators use individual module models which relate to distinct physical processes (or items of equipment) linked together according to the flowsheet topology.

1.2.1 Equation Oriented Flowsheeting

The equation based approach has a number of advantages, in particular relating to optimisation of the flowsheet. Because the flowsheet is represented by a set of equations the analytical expression for the optimisation problem is readily available. This means that any general rigorous global optimisation approach can, in principle, be applied. The availability of the equations allows reformulation into the standard forms to which rigorous global optimisation algorithms are applicable.

1.2.2 Modular Flowsheeting

Process Engineers have long thought of process models in terms of the physical systems they represent. There are many common features between plants, in particular every plant consists of a number of basic unit operations. What distinguishes one plant from another is the manner in which the units are connected and the chemical species that are processed. Consequently the most common approach to plant-wide flowsheeting is the modular approach where unit operations are modeled as procedures [1].

The unit models are typically 'black-box' models such that the procedures contained in them are not available to the flowsheeting 'executive' which solves and/or optimises the flowsheet [2]. Though the majority of rigorous global optimisation algorithms can be applied to the equation based flowsheet there has not been any work in applying rigorous global optimisation to modular flowsheets, or other modular systems.

1.3 Thesis Overview

Chapters 2 and 3 review techniques available for solving general global optimisation problems. The first chapter considers the nonrigorous global optimisation methods and the second reviews the rigorous methods.

Chapter 4 addresses the development of a global optimisation approach based on interval analysis which is suitable for solving the constrained optimisation problems which arise in Process Engineering design. The approach is flexible enough to allow a wide range of problems to be solved but can efficiently solve problems where additional information about structure and properties of the problem are available.

In chapter 5 we develop a modular flowsheeting approach which is suitable for rigorous global optimisation. The approach develops unit modules which use Extended Arithmetic Types to allow construction of modular flowsheets which provide the

information necessary to apply global optimisation. The global optimisation algorithms from chapter 4 are applied to the modular flowsheet to determine the optimal operating conditions.

Chapter 2

Nonrigorous Global Optimisation

These methods provide the capability to solve nonconvex problems where local algorithms may fail, that is: no assumptions about the convexity of the problem are made in the formulation. This does not mean, however, that the global solution will be found. Some also provide the possibility of locating a lower minimum than that produced by convex optimisation. The important advantage over global optimisation is that they can make fewer assumptions about the problem allowing a wider range of problems to be attempted. The disadvantage is simple; the solution obtained cannot be known, or proven, to be the global minimum.

The first group of methods, the Random Search techniques use some element of randomness to distribute points in the feasible region and, in principle, avoid the problem of dependency on starting point. The second section, Generalised Descent, deals with methods of changing the trajectory of local search methods to reduce the chances of converging to a local minimiser.

2.1 Random Search Optimisation

Random Search methods are characterised by the generation of a random sample in the feasible region. They are not rigorous and cannot guarantee the solution in any

more than a probabilistic sense but they can be applied to a very broad range of problems with general mathematical properties which may be intractable for other methods.

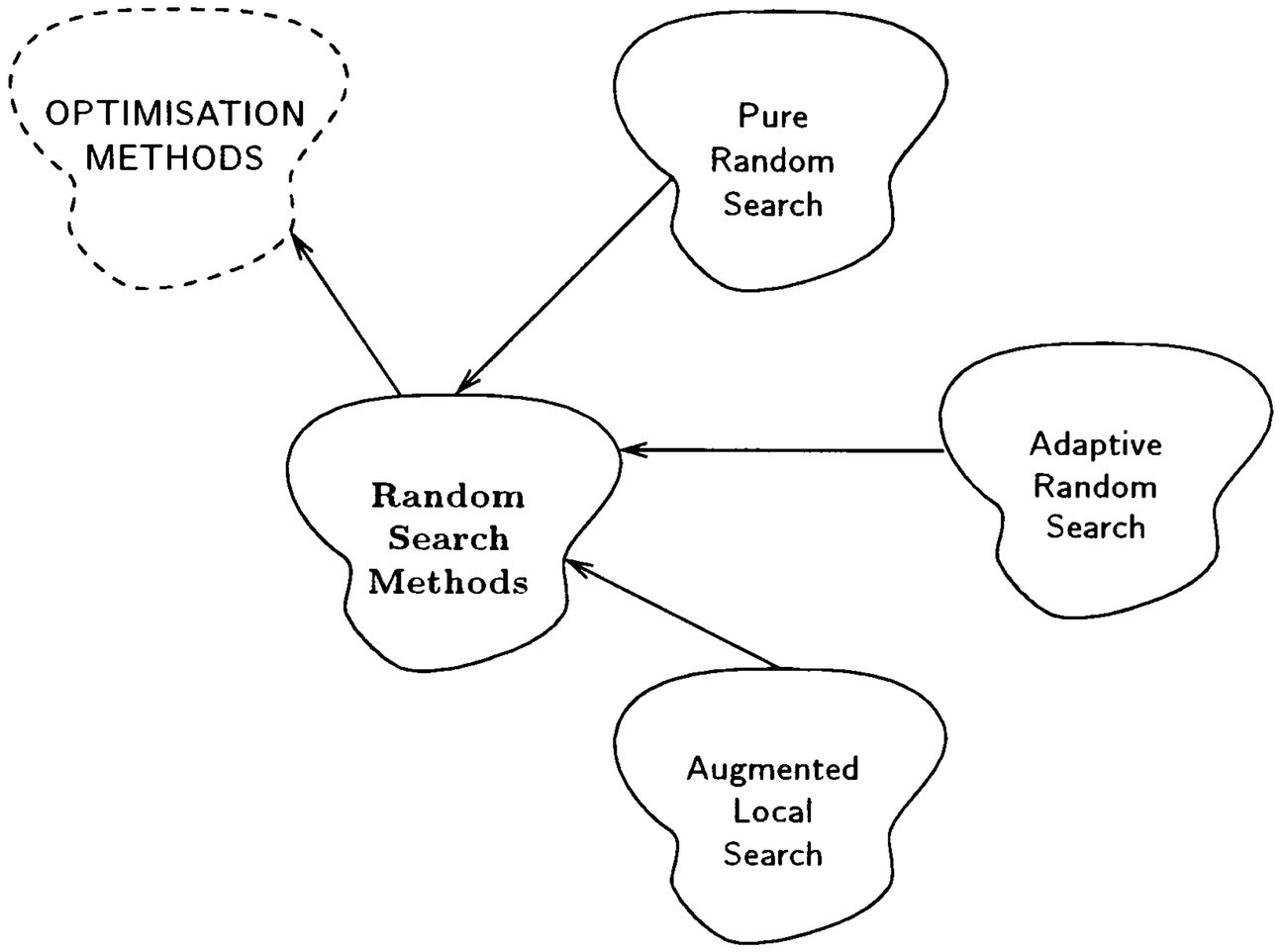


Figure 2.1: The Random Search Methods

The Random Search generates N trial points in the feasible region, $\mathcal{A}$, in a random manner. This often means distributing points x^k, $k = 1, \ldots N$, randomly in a hyper-rectangle which subscribes $\mathcal{A}$, and taking those points that lie in $\mathcal{A}$ as trial points[3]. This makes it difficult to use random search methods for solving equality constrained optimisation problems. Equality constraints are, generally, treated by the use of a penalty function.

Any of the Random Search methods may fail to locate the global minimiser if the initial search is not dense enough. It is not easy to know how dense the pattern should be in order to locate the minimiser, x^*. Further, it is not possible to make the search dense enough to guarantee that the solution is globally optimal.

Pure random methods use the random points alone, clustering and genetic searches perform further adaptive searches from the initial sample and the 'augmented lo-

cal' methods use the sample to obtain starting points for other (usually convex) optimisation algorithms.

2.1.1 Pure Random Search

Crude Sampling, the simplest of the methods, generates sample points uniformly randomly in $\mathcal{A}$. The best point attained is the algorithm's estimate of the solution to NCP.

The probability of finding the global minimum approaches unity as the number of sample points tends to infinity. This is not an efficient way of locating the global minimum but it does not require any information beyond a method of evaluating the objective function at each point and it makes no assumptions about the form or properties of the problem[3].

2.1.2 Augmented Local Search

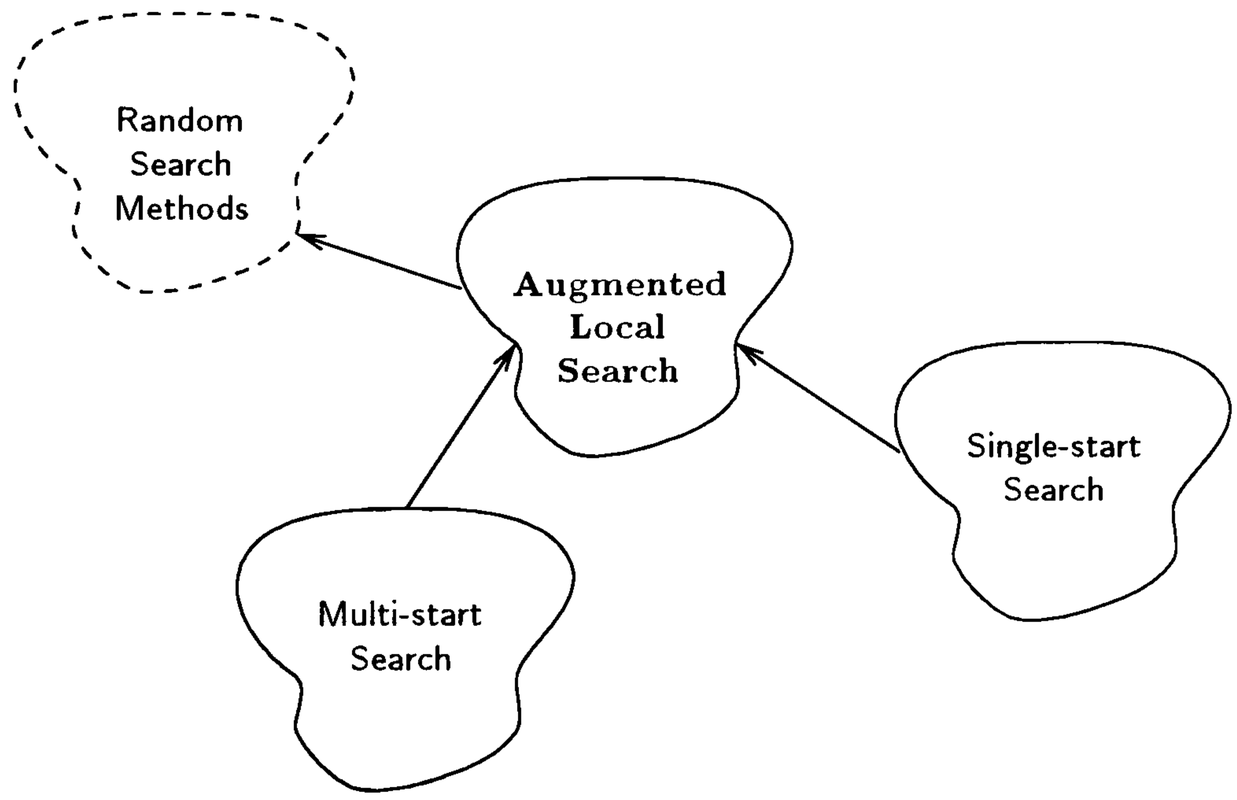


Figure 2.2: Augmenting Local Search

These algorithms use random samples in conjunction with local (convex) optimisation algorithms to improve the chances of locating a global minimiser. This com-

bination approach has an advantage over the Pure Random algorithms in that it can, typically, ensure a local solution to NCP where Pure Random search can not. The inclusion of a local search step also allows the solution of equality constrained problems. However, the class of problems to which these techniques can be applied is limited by the properties of the local method used which will be more restrictive than that of a pure search.

2.1.2.1 Single Start Random Search

Single Start random search uses the random sample to locate a good starting point for a local optimisation algorithm. It achieves this by choosing the point with the lowest objective function value. The idea behind this is to locate a point close to the global optimum from which the local phase will converge.

2.1.2.2 Multistart Random Search

A Multistart search increases the probability of locating x^* by applying a local search to each of the trial points from the initial random search phase. This assumes that $x^k \in Q(x^*)$ for some k. The difficulty is to produce the random points to a density such that this assumption is satisfied but to avoid locating the same minimum more than once. Because any number of the local searches employed may converge to the same minimiser, the Multistart algorithm, while being more likely to determine the global solution to NCP than Single Start, may be less efficient.

2.1.3 Adaptive Random Search

Random search is a very robust way of covering a surface with trial points but Crude Sampling does not, necessarily, use the available information to best advantage. It seems reasonable to distribute the sample points non-uniformly over $\mathcal{A}$, concentrating the search in more promising areas in order to improve the accuracy obtained from the same size sample.

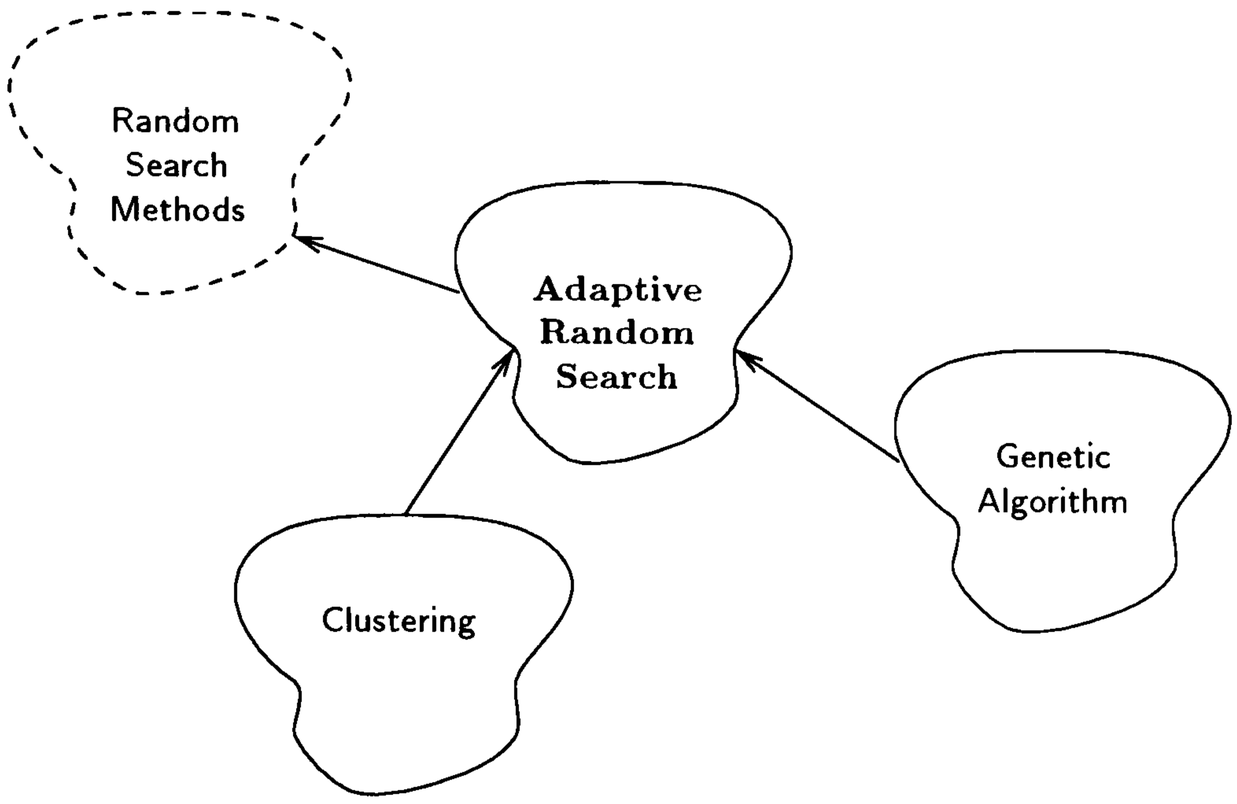


Figure 2.3: Adaptive Random Search

These algorithms use the sample to evaluate the objective function, $f(x)$, in a sequential manner modifying the sample to improve the probability of locating a global minimiser.

A wide range of heuristic techniques for adaptive Random Search have been proposed. The most important and widely used are Genetic and Clustering algorithms.

2.1.3.1 Clustering

Clustering is a method for forming clusters of points from an initially random distribution of points, $x^k, k = 1 \ldots N$. It is necessary to choose a threshold distance, δ say, below which a point is a member of a particular cluster and a definition for the distance between two points, $D(x^1, x^2)$, for example the Euclidean distance.

Clusters are grown around 'seed points', adding nearby points if $D(x_{seed}, x^k) \leq \delta$. The cluster is formed when no more points can be added.

In optimisation, extra information available allows points to be ranked so that seed points can be chosen from the points most likely to be a local minimiser[4] i.e. those with the lowest objective value.

The clustering phase is used to concentrate the sample points around local minima giving clusters of points which identify the minimisers. The algorithm can stop at this stage or can be used as the start of an Augmented Local method by applying a local optimisation from one point in each cluster to locate local minima. This has the advantages of Multistart Random Search but without the overlap[3].

2.1.3.2 Genetic Algorithms

Genetic Algorithms are a form of adaptive random search analogous to the process of natural selection in biological evolution[5]. Starting with a random sample each candidate point is represented by a 'string' analogous to a string of genes. The strings of the current sample (population) are manipulated to produce a new population.

Each population is, typically, subjected to a fitness criterion such as objective function value and the population modified appropriately. A popular choice is 'fitness proportionate reproduction' where the probability that a given point will participate in the following generation is proportional to its fitness.

New generations are formed by the application of 'genetic operators' such as the Crossover operator and Mutation operator. The Crossover operator is used to share information between generations by mating strings at random. Mating is simulated by splicing two strings at a random length. This preserves some of the characteristics of the parents in the following generation [6].

The mutation operation is used to introduce new characteristics by the occasional, random alteration of a string position.

Genetic algorithms have been applied to a wide range of problems. They are relatively simple to implement and require only a method of evaluating the objective function at each sample point. They appear to be particularly suited to solving problems of high dimensionality[7].

2.2 Generalised Descent

Generalised Descent can be thought of as an extension of convex/descent optimisation to nonconvex problems. These algorithms do not provide rigorous global optimisation. They locate local minimisers but cannot guarantee that the results are global optima. The term generalised descent covers two main classes of algo-

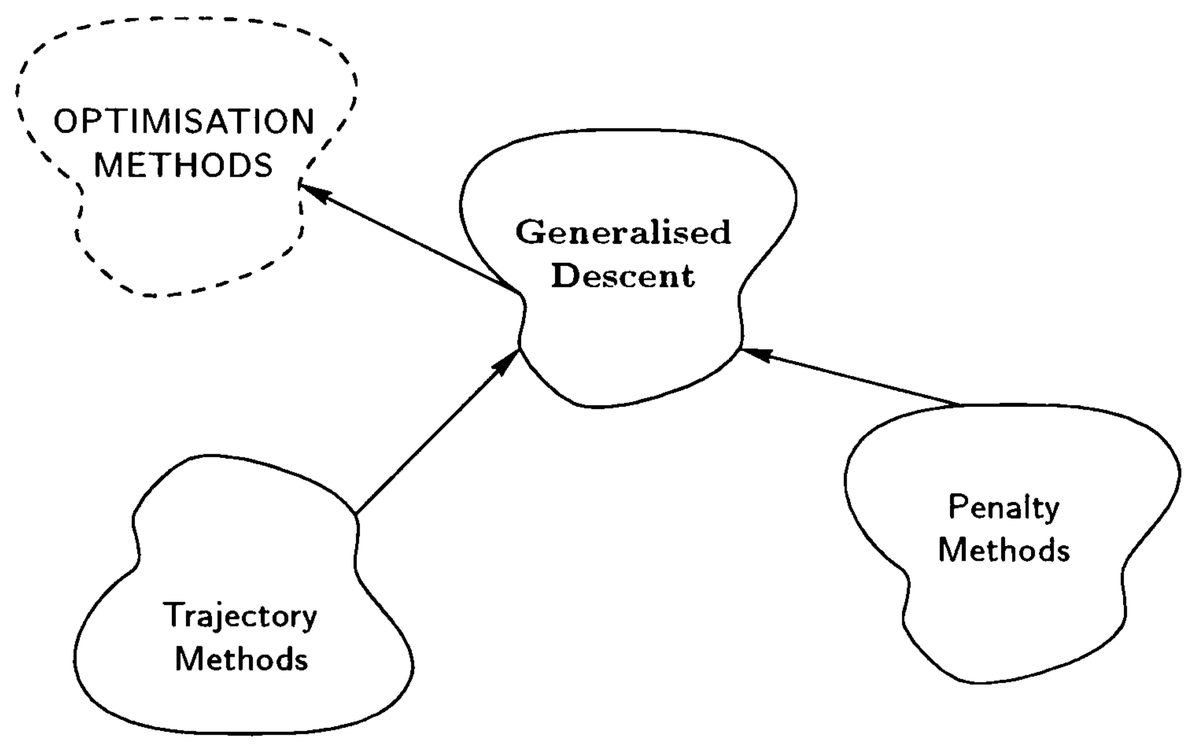


Figure 2.4: Generalised Descent Methods

rithm; the trajectory methods and the penalty methods. Both are based on the use of convex optimisation methods to solve nonconvex problems by modification of the local search and objective function respectively. The range of problems that can be solved with these methods is usually determined by the local optimisation method employed.

2.2.1 Trajectory Methods

These methods are based on modified search trajectories for local methods preventing them, in theory, from terminating at local minima and continuing to locate further minima. This is achieved by a modification of the differential equations governing the descent (and possibly ascent) of the search.

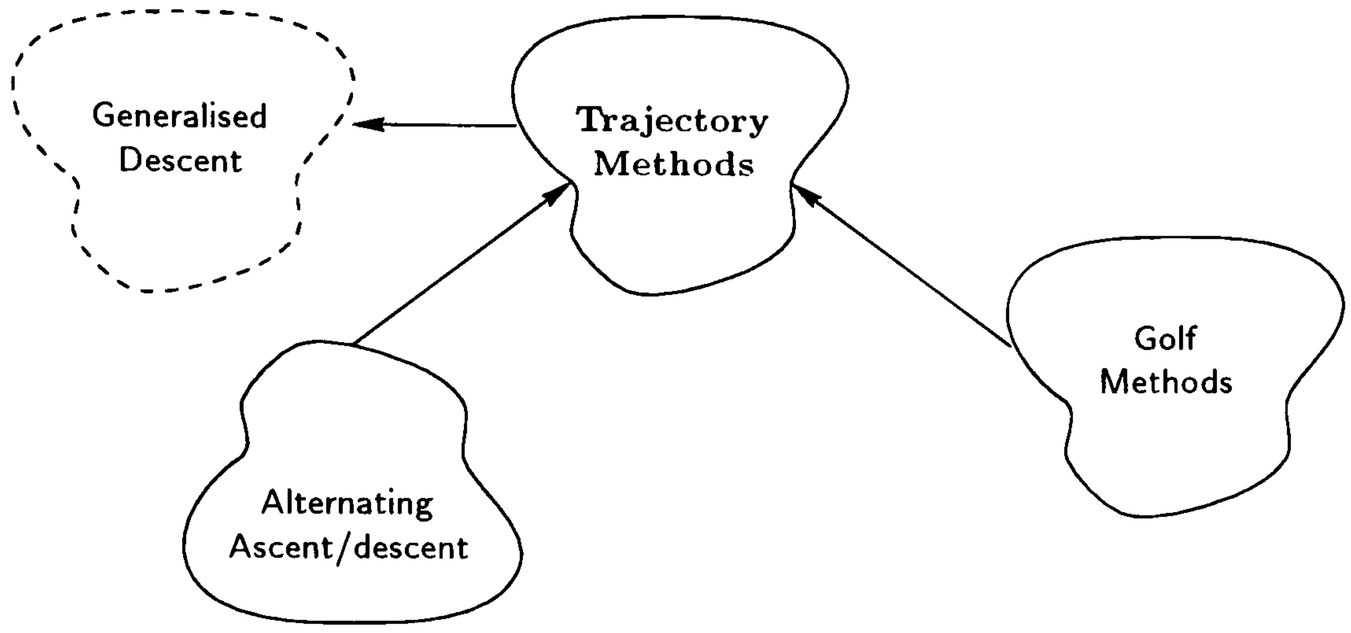


Figure 2.5: Trajectory Methods

The simplest of the trajectory methods is an alternating ascent/descent whereby a local minimum is located in the usual fashion but then the search switches to the location of a saddle point, using the largest eigenvector of $\nabla^2 f(x)$ as a direction. Once a saddle point has been located the method reverts to minimisation. In the event that one minimum be determined twice a new starting point is chosen.

Another approach, that of the golf methods, is to treat the search as a particle in a field. Providing the particle with a mass results in inertia which may cause the search to move through a shallow minimum. This is achieved by adding a second order term to the differential equation describing the position, $x(t)$. Giving

$$m(t)\ddot{x} - n(t)\dot{x} = -\nabla f(x), \tag{2.1}$$

where $m(t) \geq 0$ is the mass of the particle moving in a field $f(x)$ subject to a dissipative force $n(t) \geq 0$.

The idea is that the momentum will carry the particle through shallow minima and it will converge to a global minimiser. Because the algorithm tends to accelerate when going downhill and slows going uphill it is possible for it to converge to a small dent at the top of a hill [8].

2.2.2 Penalty Methods

Penalty methods for global optimisation are very similar to the penalty methods used in constrained convex optimisation. Rather than penalise the algorithm for violating the constraints, these methods use a penalty to avoid local minima. This is typically used within a multistart framework whereby the objective function is modified as each local minima is located to prevent the algorithm from converging to the same minimiser twice.

The choice of penalty function is based on the need to avoid introducing 'false' minima whilst keeping the penalty large enough. A range of penalty functions have been proposed for algebraic functions and polynomials. Two general approaches are the Filled Function penalty function and the Tunnelling approach.

2.2.2.1 Filled Functions

A class of penalty methods, first proposed by Ge [9][10], are referred to as 'filled function' methods. The penalty function fills the region of attraction of a minimum at the point x^k by a function with a maximum at x^k. The filled function proposed is

$$P(x, \alpha, \beta) = \frac{1}{\alpha + f(x)} \exp\left(-\frac{\|x - x^k\|^2}{\beta^2}\right) \tag{2.2}$$

The most obvious drawback of this 'filled function' scheme is its dependence on the parameters α and β which may require tuning for each problem depending on the scaling and nature of the objective function. Some standard test problems of low dimensionality are solved using the algorithm demonstrating efficiency when compared with other penalty methods [10, 11].

2.2.2.2 Tunneling

The idea behind the tunnelling algorithms is to locate an approximation to the global optimum (a local minimum), x^k, and then locate a starting point, z^k, from

which a better minimiser can be located. This is achieved by a local phase, using a local minimisation algorithm, and a tunnelling phase.

The tunnelling phase minimises an augmented objective function[12]

$$T^k(x, x^k) = \frac{f(x) - f(x^k)}{((x - x^k)^2)^\alpha} \tag{2.3}$$

to provide a new point z^k where $f(z^k) < f(x^k)$. A local search from z^k should result in a lower minimum. Then $x^{k+1} = z^k$ and the procedure is repeated. If $f(z^k) > f(x^k)$ then the parameter α is increased. The method is extended to the multidimensional case by Levy and Montalvo[13].

The advantage of the tunnelling method over the normal penalty function is that the penalty function is not minimised to find the actual minimum but, rather, to locate a new starting point. This means that minima introduced by the penalty cannot be part of the solution set because the local search from such a starting point will step away from it. The tunnelling procedure may miss minima that are close together.

2.3 Summary

The nonrigorous methods described in this chapter are frequently described in the literature as 'global' optimisation methods. These methods do not guarantee global optimality and so they are not strictly global optimisation methods at all. However, the statement is justified in some respects because they are reported to locate better optima than local algorithms and it is reasonable to assume that these solutions are frequently global minimisers[1].

The nonrigorous methods can be applied to a wider class of problems at less computation expense than rigorous methods. This is because the difficulty of global optimisation is not in *finding* the solution but in proving that the solutions are global.

[1] the augmented local methods in particular

Thus the decision to use rigorous or nonrigorous methods depends on the importance of proving that the solution is global balanced against the cost of doing so. In some cases this decision is made by the problem; for example if problems are given by a black-box routine then rigorous methods cannot be applied and a nonrigorous method must be used. Equally, for some applications, such as molecular conformation and some integer problems, the only satisfactory solution is the global solution and the problem is only solved if the global solution is definitely found, local solutions do not provide the answer to the conformation problem.

The choice then will, typically, be made on a cost basis. If it is possible to apply rigorous global optimisation to process engineering problems and the cost of doing so is not too much higher than that of nonrigorous methods then the rigorous approach is a clear winner because it will *definitely* result in global solution.

If it is practical to apply rigorous methods then they are definitely the better solution. This is why this work considers the problem of rigorous global optimisation and any reference to 'global optimisation' in the remainder of this text is meant in the strict sense where the algorithms *prove* that the solutions are global optimisers.

Chapter 3

Rigorous Global Optimisation

In order to ensure that the solution to a general nonconvex optimisation problem is global it is necessary either to locate all the minima or to cover the region of interest so that no minima are missed. These 'Covering Methods' are usually based on excluding subregions until a region, or set of regions, that is sufficiently small may be said to contain the global minimiser. To exhaustively search a region it is necessary to have some mechanism for obtaining lower, and perhaps upper, bounds on the problem over this region and an upper bound, $\overline{y^*}$, on the value of $f(x^*)$.

The algorithms rely on bounding the objective function over subsets, $\mathcal{X}^k$, of the feasible region, and maintaining an upper bound on the value of the global optimum. If the lower bound for a given $\mathcal{X}^k$ is greater than $\overline{y^*}$ then $\mathcal{X}^k$ cannot contain a global minimiser. As partitions are refined the lower bounds become tighter and form a closer approximation of the objective. When the lower bound 'meets'[1] the upper bound then a global minimiser is located in the current partition.

These methods can all be considered in a branch and bound framework, similar to that employed by some MINLP algorithms [14], where a lower bound on y^* is refined by partitioning the feasible region and getting better bounds over these regions.

[1] Different definitions of 'meets' can be used. At this point it is not important which one is used.

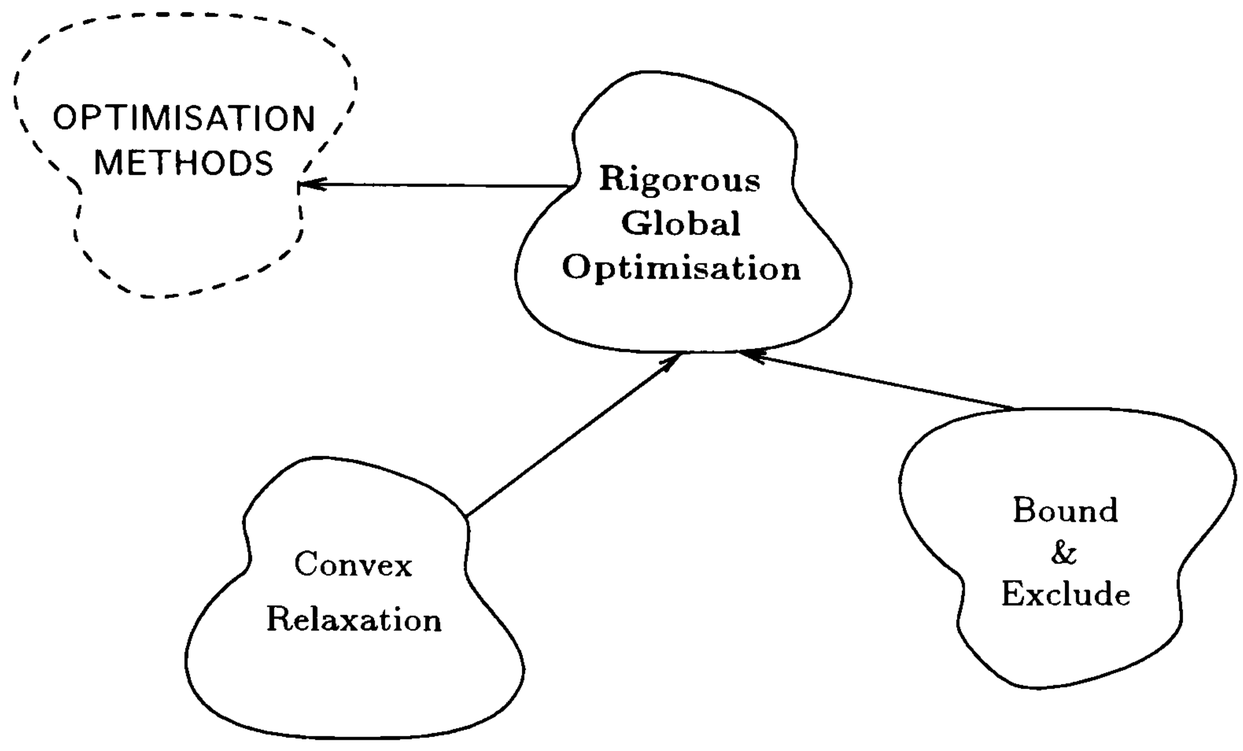


Figure 3.1: Rigorous Optimisation Methods

Different algorithms have different procedures for getting bounds, partitioning and for termination.

3.1 The Form of the Covering Algorithm

A general form of the algorithm for a Covering/Exclusion method is:

1. Initialise :
   (a) an upper bound on the solution to NCP, $\overline{y^*} = \infty$.
   (b) the initial region, $\mathcal{X}^0 \supseteq \mathcal{A}$.
   (c) a list, L, with the initial pair $(\mathcal{X}^0, \underline{y^*})$.
2. Select a region from the list, $\mathcal{X}$ and split it into subsets $\mathcal{X}^k$.
3. For each subset $\mathcal{X}^k$
   (a) Obtain a lower bound, $\underline{y^{*k}}$ on the solution to the problem subject to $x \in \mathcal{X}^k$.

(b) Add the pair $\left(\mathcal{X}^k, \underline{y^{*k}}\right)$ to the list.

(c) Obtain an upper bound, $\overline{y^*}^k$, on the value of the global minimum and set $\overline{y^*} = \min(\overline{y^*}, \overline{y^*}^k)$.

4. Discard any pairs from L, for which $\underline{y^{*k}} > \overline{y^*}$. Also apply other tests to remove pairs which cannot contain global minimisers.

5. If termination criteria apply, Terminate.

6. Go to step 2.

Certain problems can be solved more easily than this algorithm suggests by exploiting special properties of the problem. For example, minimisation of a concave function over a convex set (concave programming) is simplified because the global minimiser lies at an extreme point. In general, however, all solution methods for nonconvex programming problems can be viewed in the form outlined above. The methods differ, primarily, in the way the bounds and partitions are generated. Beyond this, a number of accelerating procedures have been proposed for most of the bounding methods.

This chapter details the main classes of global optimisation approaches which are applicable to general problems. Discussion of methods for specific classes of problem (such as concave minimisation) is not included. For each class of methods the theoretical developments are described and then the method of obtaining upper/lower bounds and the partitioning method are described.

The chapter starts with 'Relaxation Methods' which relax the nonconvex optimisation problem to another, convex, problem. Then the 'bound and exclude' methods are considered.

3.2 Relaxation Methods

One way of obtaining a lower bound on the solution to NCP over a region, $\mathcal{X}^k$, is to solve a relaxation of the original problem which underestimates NCP. By construct-

ing a relaxation of the original problem, where the relaxation can be solved globally, a lower bound can be obtained on $f(x^*)$. The solution of these underestimating (relaxed) subproblems provides a valid lower bound on NCP in a given region $\mathcal{X}^k$. Equally, a relaxed problem which can, itself, be underestimated will provide valid bounds.

An upper bound, $\overline{y^*}$, can be obtained by sampling any feasible point in $\mathcal{X}^k$ or by solving the original problem with a convex optimisation algorithm.

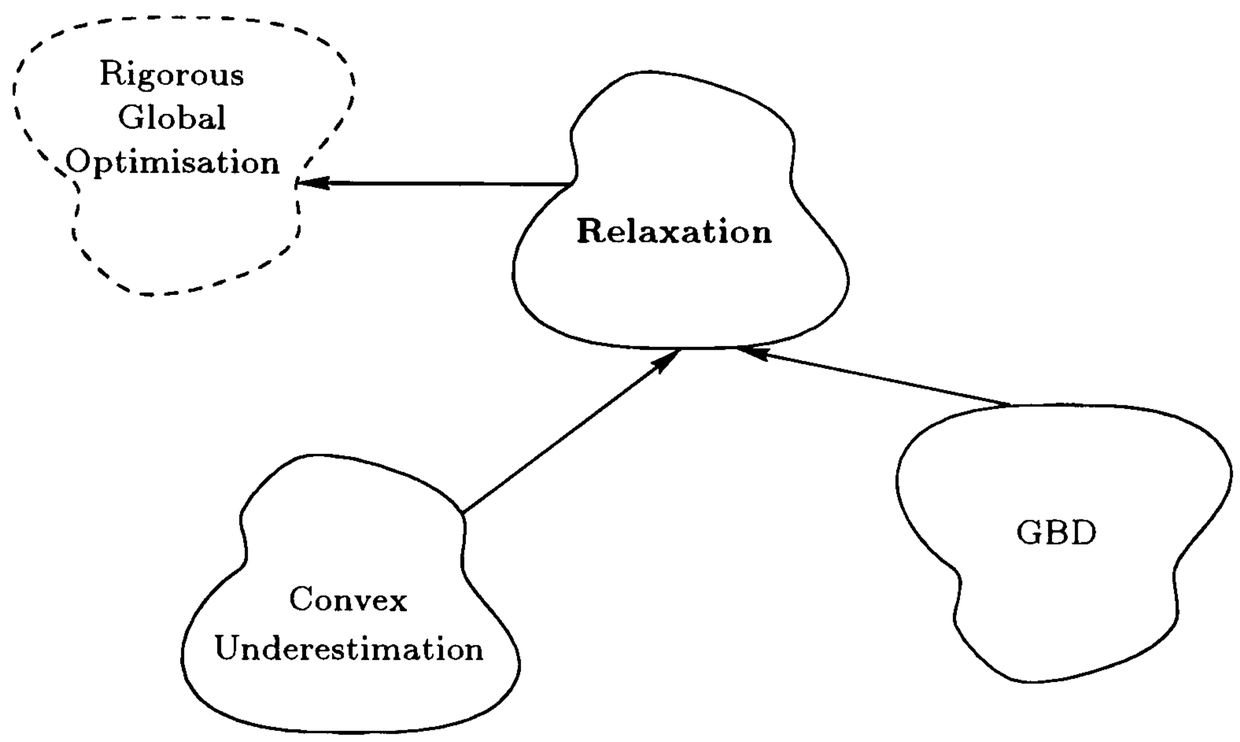


Figure 3.2: The Relaxation Methods

The algorithms generate a sequence of convex subproblems, P^k, over different partitions, $\mathcal{X}^k$, of the feasible region. The solutions to these provide a lower bound on the solution to NCP. The algorithm terminates when the upper and lower sequences 'meet' or when the minimiser of problem P^k is feasible for problem NCP and obtains the same minimum value.

The general algorithm in §3.1 includes a step for excluding regions from the search. Algorithms which solve a relaxed problem do not need to, explicitly, include such a step[15]. This is because the partition that is chosen at each iteration is the one with the lowest lower bound, a depth-first search. Thus, regions which would be deleted on the basis of lower bound are not investigated which has the same effect as deleting them – except that the memory used to store the partition is never reclaimed.

This chapter describes two important classes of underestimator. The first based on the Generalised Benders Decomposition [16] and used in the GOP algorithm[17] and the second, a more general class of convex underestimators, which have been used in a number of algorithms.

These underestimators are all restricted to certain mathematical problem types. In general transformations can be applied to a wider range of problems to produce the type required. This usually requires the addition of new variables and constraints to the original problem.

3.2.1 Generalised Benders Decomposition

The Generalised Benders Decomposition (GBD) is used to decompose problems of the form

$$\begin{aligned} \min_{v,w} \quad & f(v,w) \\ \text{s.t.} \quad & \\ g(v,w) \quad & \leq \quad 0 \\ v \quad & \in \quad V \\ w \quad & \in \quad W \end{aligned} \tag{3.1}$$

where $f(v,w)$ and $g(v,w)$ are continuous and differentiable and V and W bound the v and w variables respectively. This is called the 'outer' problem.

It is assumed that w is a vector of 'complicating' variables such that temporarily fixing them ($w = w^k$) renders a more tractable optimisation problem in the v variables. This is the case if w are integer variables and fixing them results in a continuous problem or if the problem is convex in v but not in v *and* w such that fixing the w variables results in a convex optimisation problem.

The key step is the projection of the problem onto the w-space [16] giving

$$\min_{w} \eta(w) \quad \text{s.t.} \quad w \in W \cap \mathcal{N}, \tag{3.2}$$

where

$$\eta(w) = \inf_{v \in V} [f(v, w) \quad \text{s.t.} \quad g(v, w) \leq 0] \tag{3.3}$$

and

$$\mathcal{N} = \{w | g(v, w) \leq 0 \text{ for some } v \in V\}. \tag{3.4}$$

$\eta(w)$ is the optimal value of the outer problem for $w = w^k$,

$$\begin{aligned} &\min_{v \in V} \quad f(v, w^k) \\ &\text{s.t.} \\ &\qquad g(v, w^k) \leq 0. \end{aligned} \tag{GBP}$$

This is the Generalised Benders' Primal (GBP) problem. Given the assumption that it is substantially easier to solve GBP than it is to solve the outer problem, it is important to consider using the projected problem, (3.2), which contains GBP as an inner optimisation problem, as a route to solving the outer problem.

Geoffrion [16] showed that the outer problem is infeasible or unbounded if, and only if, the same is true of the projection. Further, if w^* is optimal in the projection and v^* reaches infimum when $w = w^*$ then (v^*, w^*) is optimal in the outer problem.

The function η and the set $\mathcal{N}$ are only known implicitly making the projected problem difficult to solve in all but the simplest cases but the set $\mathcal{N}$ can also be written as the intersection of the collection of regions which contain it. This gives the (infinite) system:

$$\left[\inf_{v \in V} \psi^T g(v, w)\right] \leq 0, \quad \forall \psi \in \Psi, \tag{3.5}$$

where

$$\Psi = \left\{\psi \in \mathbb{R}^n | \psi_i \geq 0, \sum_{i=1}^{n} \psi_i = 1\right\}. \tag{3.6}$$

Using this rewritten $\mathcal{N}$ it is possible to write the projected problem (3.2) as

$$\begin{aligned} &\min_{w \in W} \quad \eta(w) = \inf_{v \in V} [f(v,w) \quad \text{s.t.} \quad g(v,w) \leq 0] \\ &\text{s.t.} \\ &\left[\inf_{v \in V} \psi^T g(v,w)\right] \leq 0, \quad \forall \psi \in \Psi, \end{aligned} \tag{3.7}$$

where Ψ is defined in (3.6). A further transformation is applied by invoking the Lagrangian dual of the inner optimisation problem. This gives an equivalent problem,

$$\min_{w \in W} \left[\sup_{\mu \geq 0} \left\{\inf_{v \in V} [f(v,w) + \mu^T g(v,w) \quad \text{s.t.} \quad (3.5)]\right\}\right]. \tag{3.8}$$

Using the definition of supremum gives the Generalised Benders' Master problem (GBM):

$$\begin{aligned} &\min_{w \in W} \quad a \\ &\text{s.t.} \\ &a \geq \inf_{v \in V} f(v,w) + \mu^T g(v,w) \qquad \text{(GBM)} \\ &\left[\inf_{v \in V} \psi^T g(v,w)\right] \leq 0, \quad \forall \psi \in \Psi, \end{aligned}$$

where a is a scalar. This problem is *equivalent* to (3.1).

Problem GBM has an infinite number of constraints (a semi-infinite programming problem). One way to solve it is by relaxation. All but a few constraints are dropped (to give an outer approximation of $\mathcal{N}$) the problem is solved and any constraints violated by the solution are added. The resulting problem is solved and any constraints violated by this solution are added. This sequence is repeated until a feasible solution is obtained.

Geoffrion's property 'P' states that the solution to the inner optimisation problem can be found independently of w. In this case the master problem can be solved and provides a lower bound on the solution of the outer problem.

If property P does not hold an optimal value and multipliers from the primal problem can be used to fix the value of the inner optimisation problem [18]. However, it is important to note that a is a valid lower bound on the solution of the outer problem if and *only* if a is the global minimum. This implicitly assumes that problem GBM is convex. When this assumption does not hold the algorithm can terminate at points which are not even local extrema [19]. Further, Sahinidis and Grossmann [20] showed that "if the problem in the xy-space [vw-space] possesses several local minima then the projection in the y-space [w-space] also has multiple local minima".

The GOP algorithm overcomes this difficulty by further relaxing the master problem so that it can be underestimated by a sequence of linear underestimators.

3.2.1.1 The GOP Algorithm

The GOP algorithm [17] has been successfully applied in a number of areas including Molecular Conformation problems and a number of Chemical Engineering problems. It uses the GBD to obtain underestimators to problems in the form of (3.1),

$$\begin{aligned} \min_{v,w} \quad & f(v,w) \qquad v \in V, w \in W \\ \text{s.t.} \quad & \\ & g(v,w) \leq 0 \\ & h(v,w) = 0 \end{aligned} \tag{3.9}$$

where $f(v,w)$ and $g(v,w)$ are: continuous, differentiable, convex in v for fixed w and convex in w for fixed v and $h(v,w)$ is linear in v for fixed w and linear in w for fixed v.

This allows the inclusion of bilinear equality constraints, $h(v,w)$, which can be expressed as two inequalities both satisfying the conditions placed on $g(v,w)$. Floudas and Liu[21] have shown that 'a large class of smooth mathematical programming problems can be converted into the form to which the GOP algorithm applies'.

3.2.1.1.1 Upper Bounds: The primal problem Upper bounds are generated by solving the primal problem (GBP) with a local algorithm. Since the primal is the original problem solved for fixed w it is convex and the solution is a valid upper bound on NCP if w^k is a feasible point.

3.2.1.1.2 Lower Bounds: The master problem The Lagrange multipliers obtained from solution of the primal problem, GBP, at the kth iteration are used to construct a relaxation of GBM by dropping the infinite system of constraints given by (3.5),

$$\begin{aligned} &\min_{w \in W} \quad a \\ &\text{s.t.} \\ &\qquad a \geq \inf_{v \in V} L(v, w, \lambda^k, \mu^k) \end{aligned} \tag{3.10}$$

where $L(v, w, \lambda^k, \mu^k)$ is the Lagrange function formulated for the Primal problem at iteration k, λ^k and μ^k are the multipliers corresponding to the equality and inequality constraints respectively.

The Lagrange function,

$$L(v, w, \lambda, \mu) = f(v, w) + {\lambda^k}^T h(v, w) + {\mu^k}^T g(v, w), \tag{3.11}$$

is convex in v for fixed w. A linearisation, $L(v, w, \lambda^k, \mu^k)|_{v^k}$, of L at v^k is an underestimator of L for fixed w. This linearisation is an underestimating plane in the v space so the solution of

$$\min_{v \in V} L(v, w, \lambda^k, \mu^k)|_{v^k} \tag{3.12}$$

occurs at an extreme point of V.

This problem is illustrated in Figure 3.3 where $v \in \mathbb{R}$ so the solution to (3.12) lies at $\underline{v}$ if the gradient is positive, and at $\overline{v}$ if the gradient is negative. Thus (for the case

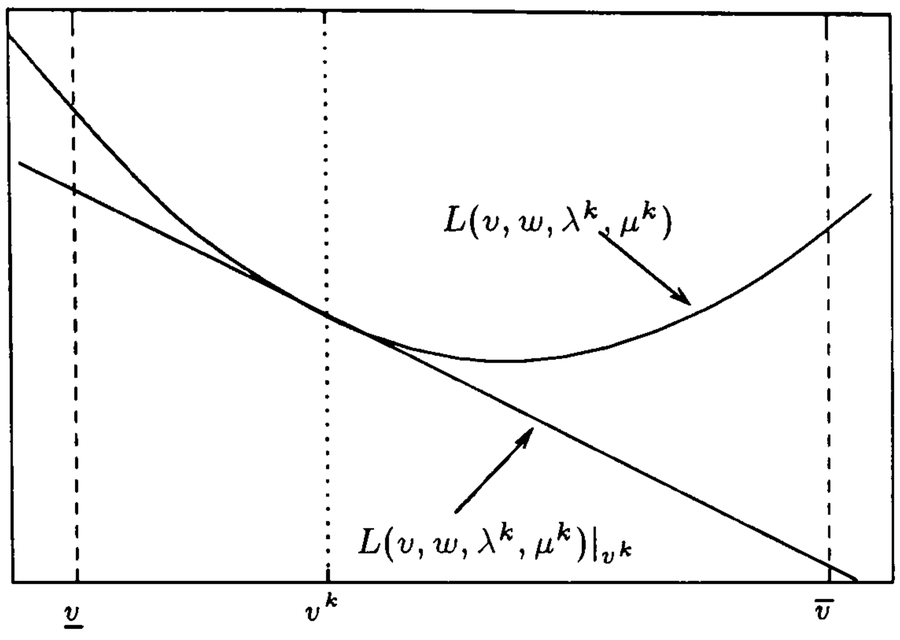


Figure 3.3: Linearised Lagrangian underestimators

shown in the figure) $L(\overline{v}, w, \lambda^k, \mu^k)|_{v^k}$ is an underestimator of $L(v, w, \lambda^k, \mu^k)$ when the gradient is negative, that is,

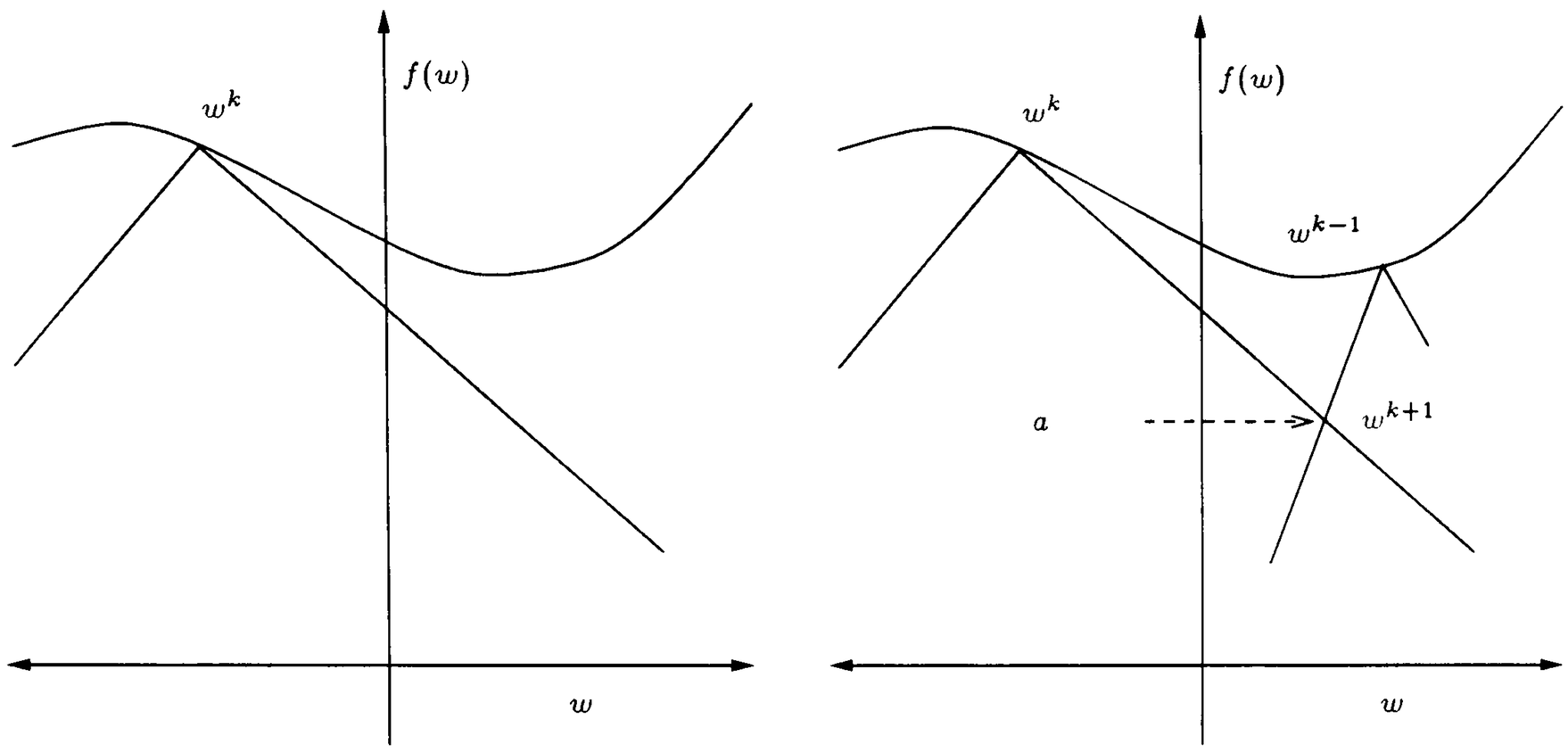
$$\nabla_v L(v, w, \lambda^k, \mu^k)|_{v^k} \leq 0. \tag{3.13}$$

This is a *qualifying* constraint.

In general, solution of an inner relaxed dual (IRD), for each set of bounds, v^b, on the v variables, provides a valid lower bound on the outer problem. Linearised Lagrangians from previous iterations can be included if the corresponding qualifying constraints are satisfied at v^k

$$\begin{aligned} \min_{w} \quad & a \\ \text{s.t.} \quad & \\ a \quad & \geq \quad L(v^b, w, \lambda^k, \mu^k)|_{v^k} \\ 0 \quad & \geq \quad \nabla_{v_i} L(v, w, \lambda^k, \mu^k)|_{v^k} \quad \text{if } v_i^b = \overline{v_i} \\ 0 \quad & \geq \; - \quad \nabla_{v_i} L(v, w, \lambda^k, \mu^k)|_{v^k} \quad \text{if } v_i^b = \underline{v_i} \end{aligned} \tag{IRD}$$

In summary; the relaxed master problem is a relaxation of the outer problem (3.9), the linearised Lagrangian underestimates the relaxed master so the solution to IRD is a valid lower bound on the outer problem.



(a) Construction of underestimator at w^k (b) Solution of the Relaxed Dual

Figure 3.4: GOP Relaxed Master Problem

The Relaxed Master is, essentially, a minimisation of the combination of underestimators active in a given plane. This is shown in Figure 3.4. The curve represents the solutions of the inner problem in w,

$$\inf_{v \in V} L(v, w, \lambda^k, \mu^k), \tag{3.14}$$

for different values of w.

3.2.1.1.3 Partitioning At each iteration the value of $w = w^k$ is fixed, the Primal problem is solved to obtain an upper bound on y^* and provide the Lagrange multipliers required to generate the underestimators. IRD is then solved with underestimators from previous iterations if their qualifying constraints are satisfied. This provides a lower bound, a, on y^* and an optimal value of $w = w^{k+1}$ which is used to fix the w variables in the next primal problem.

The qualifying constraints partition the feasible region in the w space into a partition for each set of bounds on the v variables.

The algorithm proceeds in a sequential manner refining the lower bounding functions by constructing more underestimators and adding more qualifying constraints to the inner relaxed dual. It terminates when the primal and IRD solutions "meet".

The GOP algorithm suffers from two main weaknesses. The number of subproblems which need to be solved increases exponentially with the size of the problem and the number of constraints in each subproblem increases at every iteration. Some of the problems have been addressed in new formulations of GOP [22] which use mixed integer formulations to solve one MINLP subproblem per iteration.

3.2.2 Convex Underestimation

Convex underestimation algorithms use a sequence of approximations to the convex envelope of the nonconvex problem to generate lower bounds. The convex hull and convex envelopes are defined in §A.1.3.

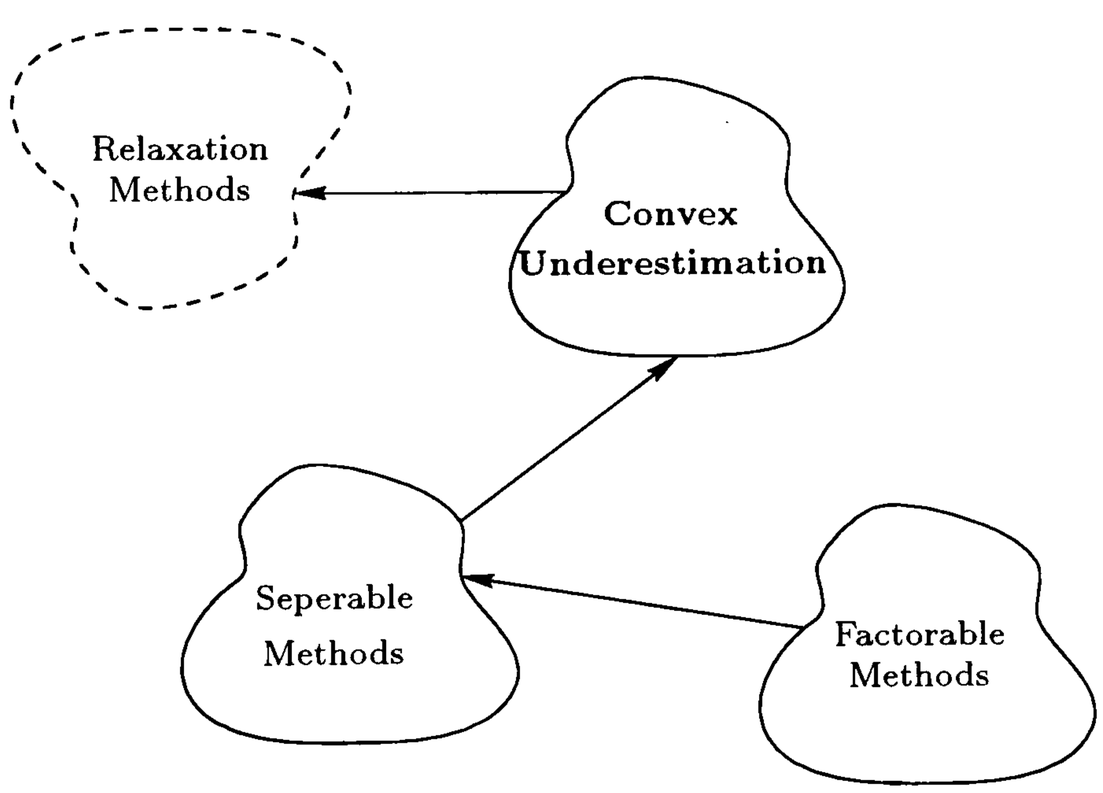


Figure 3.5: Convex Underestimation Methods

Every problem, NCP, for which $\mathcal{A}$ is convex has an associated convex problem, formulated using its convex envelope which has the same global minimiser[2]. Solution

[2]This is a corollary of the definition of the convex hull

of this problem provides the solution to NCP with one application of a convex optimisation algorithm.

As the generation of the convex envelope for a general problem can be more difficult than the global optimisation problem itself the convex underestimation algorithms use successive approximations of the convex envelope over partitions of the feasible region to solve NCP in a number of steps.

3.2.2.1 Underestimators for Separable Functions

Falk and Soland[23] proposed a global optimisation algorithm using convex envelopes for problems with a separable objective

$$\min_{x \in \mathcal{A}} \quad \sum_{i=1}^{n} f_i(x_i). \tag{SNP}$$

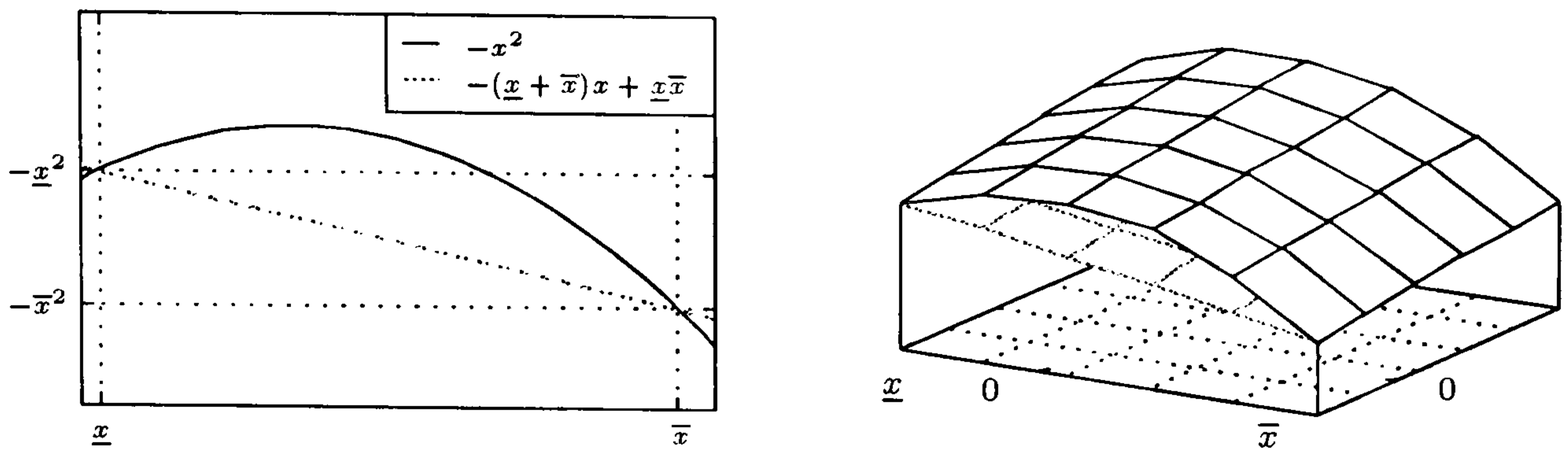
This is a separable nonconvex program (SNP) with n variables, $\mathcal{A}$ is closed and convex and each f_i is continuous on $X = \{x | \underline{x} \leq x \leq \overline{x}\}$ where $X \supseteq \mathcal{A}$.

Upper bounds on the solution to SNP can be obtained by solving the problem with a local algorithm. A lower bounding problem can be constructed by exploiting the properties of convex envelopes and separable functions.

3.2.2.1.1 Lower Bounds The convex envelope, $e_f(x)$, of a separable function, $f = \sum_i f_i(x_i)$, over a rectangular set, X, is the sum of the convex envelopes of the component functions, f_i. That is

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} f_i(x_i) \\ e_f(x) &= \sum_{i=1}^{n} e_{f_i}(x_i) \qquad x \in X. \end{aligned} \tag{3.15}$$

Figure 3.6 shows this graphically for the separable function $-x^2 + 2y$.



(a) Graph of $-x^2$ and its convex envelope

(b) Plot of $-x^2 + 2y$ and its convex envelope

Figure 3.6: The convex envelope of a separable function.

The algorithm proposed constructs a sequence of subproblems, P^k,

$$\min_{x \in \mathcal{A} \cap X^k} \quad e_f^k(x). \tag{3.16}$$

If $\mathcal{A}$ is a rectangular set then the solution to this problem is the solution to SNP. In general however, if $\mathcal{A}$ is a convex set then the global solution of problem P^k can be found by convex optimisation and is a valid underestimator of SNP on $\mathcal{A} \cap X^k$.

Soland [24] extends the algorithm to the solution of problems where the feasible region is separable and nonconvex. This is achieved by replacing $\mathcal{A}$ with a convex set, $\mathcal{S}^k$, containing the convex hull of $\mathcal{A} \cap X^k$. If the problem has only inequality constraints:

$$\mathcal{A} = \{x \in X^k | g_m(x) \leq 0\} \tag{3.17}$$

then this outer approximating convex set can be formed by replacing each active nonconvex constraint, $g_m(x)$, with its convex envelope. In this case problem P^k will have the form

$$\begin{aligned} \min_x \quad & e_f^k(x) \\ \text{s.t.} \quad & \\ & x \in \mathcal{S}^k \cap X^k \end{aligned} \tag{3.18}$$

where $\mathcal{S}^k$ is a convex set which encloses $\mathcal{A}$.

3.2.2.1.2 Partitioning If x^k, the solution to relaxed problem P^k, is feasible for SNP and

$$e_f(x^k) = f(x^k) \tag{3.19}$$

then x^k is the global minimiser, x^*, of $f(x)$ on $\mathcal{A}$.

When this does not hold and $e_f(x^k) < f(x^k)$ then the partition, X^k, must be refined. The authors propose two refining rules which they call 'strong' and 'weak'. The weak refining rule splits X^k at the solution x^k of problem P^k in direction i so as to maximise the difference

$$f_i(x_i^k) - e_{f_i}^k(x_i^k) \tag{3.20}$$

splitting X^k into $[\underline{x}^k, x_i^k]$ and $[x_i^k, \overline{x}^k]$. The strong refining rule makes the same partition but for *every* i for which

$$f_i(x_i^k) - e_{f_i}^k(x_i^k) > 0 \tag{3.21}$$

which may divide X^k into as many as 2^n rectangular subsets. These two rules are used to prove convergence of the algorithm under different conditions.

Given that it is, *in principle*, possible to 'convert just about any nonlinear programming problem into an equivalent separable programming problem'[25] this approach should be widely applicable. However, this transformation may be difficult and results in an increase in dimensionality and in the number of constraints. A more general class of problems, factorable programs, is considered by McCormick[25] who extends the approach.

3.2.2.2 Underestimators for Factorable Programs

A factorable function is one which can be written as the last function in a finite sequence of functions, $f_j : \mathbb{R}^n \to \mathbb{R}$, such that

$$f_j(x) = x_j, \qquad j = 1 \ldots n \tag{3.22}$$

and one of

$$\begin{aligned} f_j(x) &= f_p(x) + f_q(x) \\ f_j(x) &= f_p(x) - f_q(x) \\ f_j(x) &= f_p(x) f_q(x) \\ f_j(x) &= T(f_p(x)) \end{aligned} \tag{3.23}$$

where $n \leq p, q < j$ and $T : \mathbb{R} \to \mathbb{R}$.

To obtain bounding functions for factorable functions we consider the problem of obtaining convex lower bounds on the four arithmetic operations applied to $u = U(x), v = V(x)$ where $U(x)$ and $V(x)$ are functions of n variables on $X = [\underline{x}, \overline{x}]$.

The first two cases, addition and subtraction, utilise the convex/concave envelopes of $U(x), V(x)$. Given that the envelope of a separable function is the sum of the envelopes we have

$$\begin{aligned} y &= u + v, \quad e_y = e_U + e_V, \quad E_y = E_U + E_V \\ y &= u - v, \quad e_y = e_U - E_V, \quad E_y = E_U - e_V, \end{aligned} \tag{3.24}$$

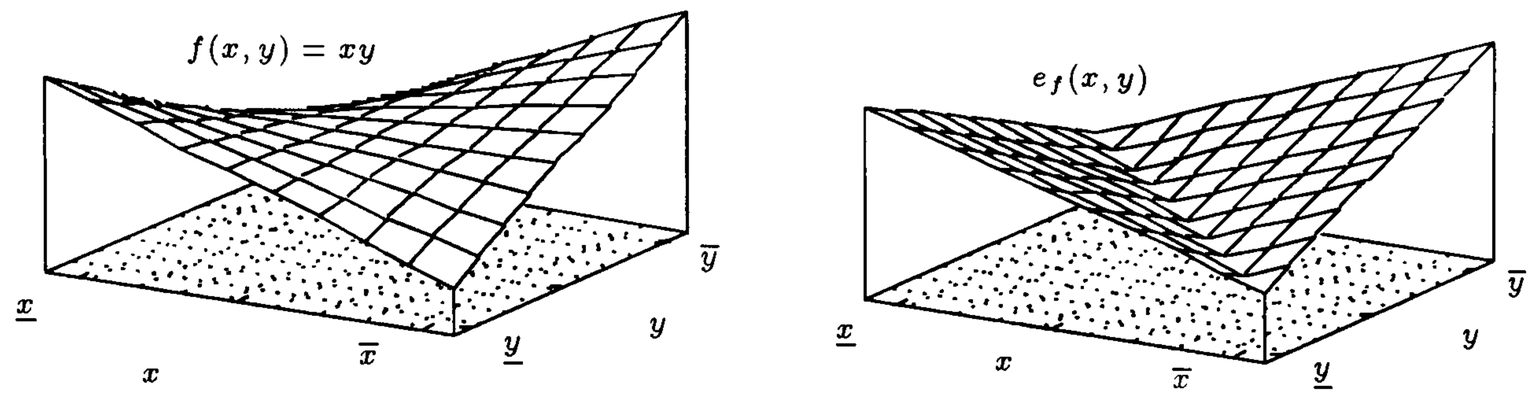
where $e_F(x)$ is the convex envelope of a function $F(x)$ and $E_F(x)$ is the concave envelope of $F(x)$. The envelopes are also functions of the interval, X, but this dependency is not made explicit to simplify the notation.

This also follows for $c_F(x)$ some convex underestimator of $F(x)$ and $C_F(x)$ a concave overestimator of $F(x)$ giving,

$$\begin{aligned} y &= u + v, \quad c_y = c_U + c_V, \quad C_y = C_U + C_V \\ y &= u - v, \quad c_y = c_U - C_V, \quad C_y = C_U - c_V. \end{aligned} \tag{3.25}$$

The next case, multiplication, is provided by McCormick [25]

$$y = uv, \quad \begin{aligned} c_y &= \max[\overline{v}c_U + \overline{u}c_V - \overline{uv}, \quad \underline{v}c_U + \underline{u}c_V - \underline{vu}] \\ C_y &= \min[\overline{v}C_U + \underline{u}C_V - \underline{u}\overline{v}, \quad \underline{v}C_U + \overline{u}C_V - \overline{u}\underline{v}] \end{aligned} \tag{3.26}$$



(a) $f(x,y) = xy$ (b) Convex Envelope of $f(x,y)$

Figure 3.7: The convex envelope of a product term

For the most basic case, $f(x,y) = xy$, this underestimator is the maximum of two intersecting planes as shown in Figure 3.7(b).

These operations require the knowledge of an under/over estimator pair and an upper and lower bound for each function $U(x)$ and $V(x)$. If these functions are composed of the arithmetic operations alone then the rules above can be applied recursively to obtain the necessary information.

The final development is to obtain similar bounding rules for terms of the form $T(z)$ where T is a function of a single variable (such as $\ln z, z^2$ or $1/z$) and $z = U(x)$. This information can be calculated for a continuous function, T, of a single variable, z, if the convex/concave envelopes of $T(z)$ on $Z = [\underline{z}, \overline{z}]$ are known and

$$\begin{aligned} \underline{t} &= T(z_{\min}) = \inf_{z \in Z} T(z) \\ \overline{t} &= T(z_{\max}) = \sup_{z \in Z} T(z), \end{aligned} \tag{3.27}$$

that is, the values of z which maximise and minimise $T(z)$ on Z are known.

McCormick[25] shows that

$$\begin{aligned} T[U(x)] &\geq e_T[\mathrm{mid}(c_U(x), C_U(x), z_{\min})] \\ T[U(x)] &\leq E_T[\mathrm{mid}(c_U(x), C_U(x), z_{\max})] \end{aligned} \tag{3.28}$$

where, as before, $c_U(x)$ is a convex underestimator for $U(x)$ and $e_T(z)$ and $E_T(z)$ are the convex and concave envelopes of $T(z)$ respectively. This assumes that it is possible to obtain the envelopes, maximiser and minimiser of $T(z)$ on the interval Z.

Using these results it is possible to apply the rules recursively to obtain upper and lower bounding functions on a factorable function $F(x)$ given by (3.23) if concave/convex over/under-estimators are known for each of the component functions $f_j(x), j > n$ and the envelopes and extreme points are known for each of the $T(z)$ forms.

3.2.2.2.1 Lower Bounds Lower bounds on the solution to NCP on an interval X^k can be obtained by solving problem P^k, a convex relaxation of NCP. This convex relaxation is obtained using the rules outlined above to derive a convex underestimator, $c_f(x)$, of the objective function and a convex relaxation of the feasible region by replacing each inequality, $g_j(x) \leq 0$ with its convex underestimator, $c_j(x)$ on X^k. This gives problem P^k

$$\begin{aligned} \min_{x \in X^k} \quad & c_f(x) \\ \text{s.t.} \quad & \\ & c_j(x) \leq 0 \end{aligned} \tag{3.29}$$

which is convex and provides a valid lower bound on NCP.

The approach to factorable functions is augmented by Epperly [26] with the development of constraints based on positive semi-definite combinations of quadratic terms. These constraints produce an exact relaxation of NCP when constructed at the global minimum which gives the algorithm finite convergence properties [27].

The class of factorable functions is quite broad and the underestimators proposed by Falk & Soland and McCormick have been used extensively. An alternative approach is to formulate a general underestimator strategy for nonconvex terms. This is the approach taken in the α-BB algorithm.

3.2.2.3 Underestimators for General Nonconvex Terms

Adjiman *et. al.*[28] extend the convex underestimating approach by proposing underestimators for general nonconvex terms using a combination of interval analysis and characteristic polynomials.

The convex underestimator for a twice continuously differentiable nonconvex term, $T(x)$ on an interval $X^k = [\underline{x}, \overline{x}]$ is given by

$$c_T(x) = T(x) + \sum_{i=1}^{n} \alpha_i (x_i - \underline{x}_i)(x_i - \overline{x}_i) \tag{3.30}$$

where

$$\alpha_i \geq \max[0, -\frac{1}{2} \min_{j, x \in X^k} \lambda_j(x)] \tag{3.31}$$

if $T(x)$ is a function of x_i and $\alpha_i = 0$ if not, the $\lambda_j(x)$ terms are the eigenvalues of $T(x)$.

Thus, $c_T(x)$ will underestimate $T(x)$ if the minimum eigenvalue of $T(x)$ can be found or a lower bound on the minimum eigenvalue. This lower bound can be obtained from the interval characteristic polynomial of $T(x)$ on X^k,

$$P_{T,X^k}(\lambda) \quad = \quad \det[\nabla^2 T(X^k) - \lambda \mathbf{I}] \quad = \quad \sum_{i=0}^{n-1} (A_i(x)\lambda^i) + \lambda^n \tag{3.32}$$

where the coefficients of λ are intervals, $A_i = [\underline{a}_i, \overline{a}_i]$, and $\mathbf{I}$ is the identity matrix.

The smallest root of the interval polynomial is smaller than the smallest root of any of the characteristic polynomials of $T(x)$ on X^k. So solution of the interval characteristic polynomial would provide a lower bound on the minimum eigenvalue,

$$\lambda_{\min} = \min_{j, x \in X^k} \lambda_j(x). \tag{3.33}$$

Solution of this interval polynomial is not a trivial task but, fortunately, it can be achieved at a fairly modest computational cost by use of the Kharitonov polynomials. The smallest root of the Kharitonov polynomials,

$$\begin{aligned}
K_1(T,X,\lambda) &= \underline{a}_0 + \underline{a}_1\lambda + \overline{a}_2\lambda^2 + \overline{a}_3\lambda^3 + \underline{a}_4\lambda^4 + \underline{a}_5\lambda^5 + \overline{a}_6\lambda^6 + \cdots \\
K_2(T,X,\lambda) &= \overline{a}_0 + \overline{a}_1\lambda + \underline{a}_2\lambda^2 + \underline{a}_3\lambda^3 + \overline{a}_4\lambda^4 + \overline{a}_5\lambda^5 + \underline{a}_6\lambda^6 + \cdots \\
K_3(T,X,\lambda) &= \overline{a}_0 + \underline{a}_1\lambda + \underline{a}_2\lambda^2 + \overline{a}_3\lambda^3 + \overline{a}_4\lambda^4 + \underline{a}_5\lambda^5 + \underline{a}_6\lambda^6 + \cdots \\
K_4(T,X,\lambda) &= \underline{a}_0 + \overline{a}_1\lambda + \overline{a}_2\lambda^2 + \underline{a}_3\lambda^3 + \underline{a}_4\lambda^4 + \overline{a}_5\lambda^5 + \overline{a}_6\lambda^6 + \cdots,
\end{aligned} \tag{3.34}$$

is less than or equal to $\lambda_{\min}$. Given $\lambda_{\min}$, the parameter α can be determined and the underestimator for the general nonconvex term constructed.

The approach taken here is applicable to any twice differentiable problem but the underestimators that can be constructed are looser for some terms than the underestimators for factorable terms. This follows from the definition of the convex envelope, the underestimators for concave and bilinear terms are the envelopes of these functions. Therefore they are the best convex underestimators possible. For this reason the algorithmic approach proposed for α-BB uses McCormick underestimators for terms of special structure and the general underestimators are applied to 'general nonconvex terms'. The underestimators proposed in this section are more generally applicable and require fewer additional variables and less reformulation.

3.3 Bound & Exclude Methods

The convex relaxation methods generate a sequence of relaxed problems to obtain lower bounds on y^*. This requires the formulation of a relaxed problem at each step and that the relaxed problem can be solved. This second requirement has an impact on the range of problems to which the convex relaxation methods can be applied. An alternative approach, which we shall call 'Bounding & Excluding', is to use a simpler, more efficient method to generate bounds on $f(x)$, $g(x)$ and $h(x)$ over each partition. As a lower bound of $f(x)$ is also a lower bound on $f(x^*)$ it can

be used as part of an exclusion strategy. Note, at this point, that the two bounds are equivalent for an unconstrained problem.

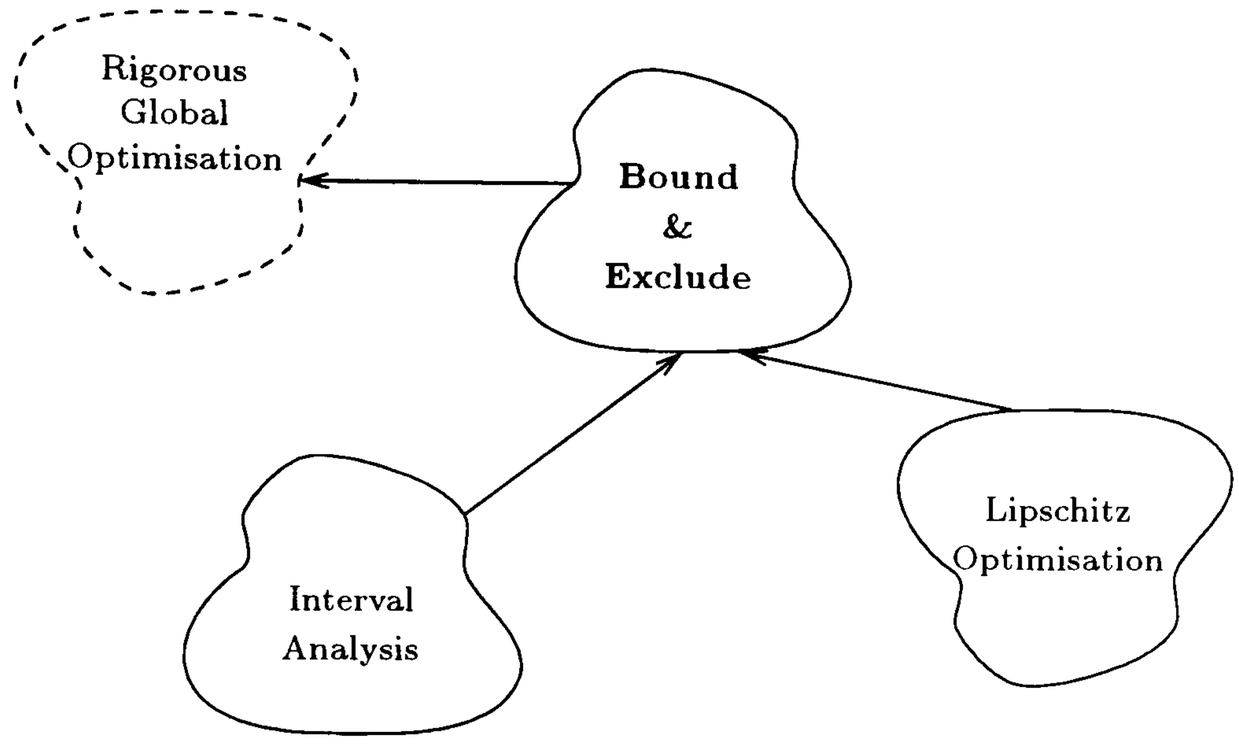


Figure 3.8: The Bound & Exclude Methods

Regions which lie outside the bounds on the constraints can be excluded. As partitions are made the bounds on $f(x)$ become tighter and the feasible region is refined.

The following sections describe the use of the Lipschitz constant and interval analysis to provide bounds on the objective over a partition.

3.3.1 Lipschitz Optimisation

It is common, and not unreasonable for practical functions, to assume that the rate of change of an objective function is bounded by some constant. This means that the region of interest can be adequately searched by the use of a grid with sufficient density[3]. If the objective function, $f : \mathbb{R} \to \mathbb{R}$ is Lipschitz with known constant, L on an interval $X = [\underline{x}, \overline{x}]$ then:

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2| \quad \forall x_1, x_2 \in X. \tag{3.35}$$

Consider then the density of the grid which will provide a solution to the required accuracy. Evaluating $f(x)$ at N points x^k in $X = [\underline{x}, \overline{x}]$, given by

$$x^k = \underline{x} + \frac{(2k-1)\epsilon}{L} \tag{3.36}$$

$$N \geq \frac{L(\overline{x} - \underline{x})}{2\epsilon} \tag{3.37}$$

will result in at least one point, x^*, satisfying the ϵ–global optimality criterion (1.3) [29].

The Lipschitz constant can also be used to construct linear lower bounding functions at any point $x^k \in X$

$$J^k(x) = f(x^k) - L\left\|x - x^k\right\|. \tag{3.38}$$

This is the approach taken by Lipschitz optimisation algorithms.

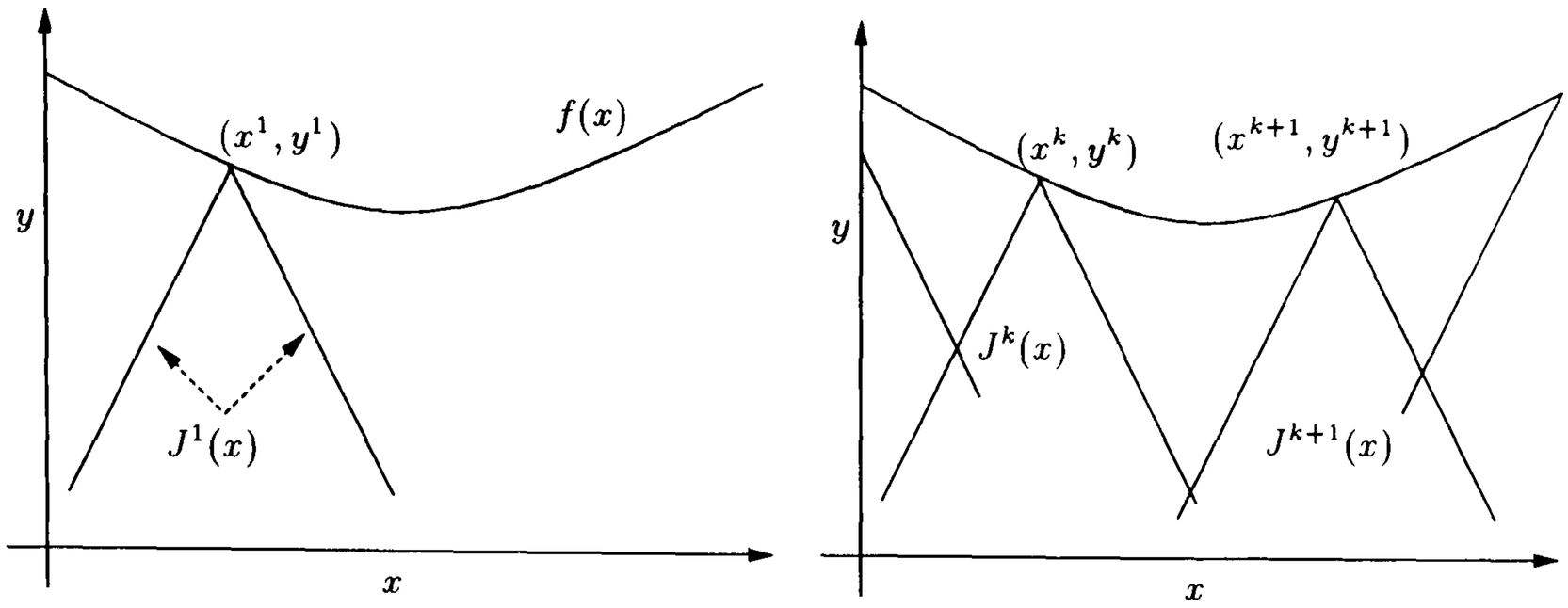
A sequential algorithm for minimising univariate Lipschitz functions proposed by Piyavskii [29] and later refined by Shubert [30] constructs a piecewise linear bounding function, $J(x)$, called a 'saw-tooth', from lower bounding functions, $J^k(x)$, at each sample point x^k given by

$$J^k(x) = f(x^k) - L|x - x^k|. \tag{3.39}$$

The construction of $J^1(x)$ is shown in Figure 3.9(a). The gradient of the underestimating function, $J^1(x)$, is the Lipschitz constant, L, of $f(x)$.

3.3.1.1 Lower Bounds

The minimum intersection point of all the underestimating functions – that is the minimum of the saw-tooth function – is a lower bound on the objective and hence



(a) Construction of $J^k(x)$ (b) The 'saw-tooth' function, $J(x)$

Figure 3.9: Lipschitz underestimators using Piyavskii's Algorithm.

NCP. The saw-tooth function, a section of which is shown in Figure 3.9(b), is given by

$$J(x) = \min_k J^k(x). \tag{3.40}$$

3.3.1.2 Upper Bound

The upper bound on the minimum, $\overline{y^*}$, is calculated from the lowest sampled point by

$$\overline{y^*} = \min_k f(x^k). \tag{3.41}$$

3.3.1.3 Partitioning

Although the algorithm according to Piyavskii [29] does not mention partitioning it is possible to interpret the algorithm as partitioning to the left and right of each sample point. Each 'partition' can be identified with an intersection point of the saw-tooth. At each step the lowest point (partition) is chosen and removed from the list of points. The underestimating function, $J^k(x)$, is constructed at the intersection producing one intersection (or partition) to the left and one to the right.

Thus each of the sets, $\mathcal{X}^k$, produced has a lower bound given by the intersection point. If this lower bound is greater than $\overline{y^*}$ then $\mathcal{X}^k$ cannot contain a global minimiser and should be discarded.

The Lipschitz methods are, in general, effective for single variable problems if $f(x)$ is known to be Lipschitz and L can be found. Complications arise when solving multivariate problems because the regions or sets which can be excluded are hyper-spheres. This is typically treated by considering simplices subscribed by the sphere.

Meewella and Mayne [31] extend the method to multi-dimensional problems by using square partitions (called cells by the authors) and constructing a set of Lipschitz underestimators for each partition.

Extensive research on the convergence properties of Lipschitz algorithms has been described in [32], [33] and [34].

3.3.2 Interval Methods

The interval algebraic system was first applied to bounding the error in finite arithmetic operations on computers [35]. An operation on a number is, instead, posed as an operation on an interval which bounds the number; the resulting interval bounds the answer. This provides a representation of the number and the absolute error incurred.

An interval is an ordered pair of real numbers $X = [\underline{x}, \overline{x}]$. It is the set of real numbers $\{x \in \mathbb{R} | \underline{x} \leq x \leq \overline{x}\}$. A degenerate interval $[a, a]$ is a real number in much the same way as a complex number $z = a + 0i$ is real. The set of all possible intervals is denoted by $\mathbb{I}$ and $\mathbb{R} \subset \mathbb{I}$.

Arithmetic operations on intervals $U = [\underline{u}, \overline{u}]$ and $V = [\underline{v}, \overline{v}]$ are defined as

$$\begin{aligned} U + V &= [\underline{u} + \underline{v}, \overline{u} + \overline{v}] \\ U - V &= [\underline{u} - \overline{v}, \overline{u} - \underline{v}] \\ U \times V &= [\min(\underline{uv}, \underline{u}\overline{v}, \overline{u}\underline{v}, \overline{uv}), \max(\underline{uv}, \underline{u}\overline{v}, \overline{u}\underline{v}, \overline{uv})] \\ U \div V &= [\underline{u}, \overline{u}] \times [1/\overline{v}, 1/\underline{v}] \quad \text{iff} \quad 0 \notin [\underline{v}, \overline{v}]. \end{aligned} \tag{3.42}$$

Further, the width of U, $w(U)$, is defined as $\overline{u} - \underline{u}$. Relational operators follow the rule; $U < V$ iff $\overline{u} < \underline{v}$. Thus the sign of U may be positive ($\underline{u} > 0$), negative ($\overline{u} < 0$) or both ($0 \in [\underline{u}, \overline{u}]$).

3.3.2.1 Inclusion Functions

The number of practical problems that interval analysis could be applied to would be limited if only those functions in (3.42) (and their combinations) could be used. Thus, the concept of an inclusion function is introduced and the properties of the different forms of inclusion function are described.

Define the range of a function $\phi : \mathbb{R} \to \mathbb{R}$ over an interval V as $\hat{\phi}(V)$.

$$\hat{\phi}(V) = \{\phi(v) | v \in V\}. \tag{3.43}$$

A function $\Phi : \mathbb{I} \to \mathbb{I}$ is an inclusion of $\phi : \mathbb{R} \to \mathbb{R}$ on $\mathcal{D}$ if [3],

$$\hat{\phi}(V) \subseteq \Phi(V), \quad \forall V \in \mathcal{D}. \tag{3.44}$$

This system is useful for global optimisation because an inclusion, $F'(X)$, of $f'(x)$, collects information about the gradient of $f(x)$ for all $x \in X$. Thus, if $F'(X) \not\ni 0$ then $f'(x) \neq 0, \forall x \in X$ and X does not contain a stationary point. This is a

[3] In the case of vector values, functions (3.44) must be satisfied componentwise

consequence of what is often called the fundamental property of interval arithmetic; that is for $\Phi : \mathbb{I} \to \mathbb{I}$ an inclusion of $\phi : \mathbb{R} \to \mathbb{R}$;

$$x \in X \Rightarrow \phi(x) \in \Phi(X), \quad (x \in \mathbb{R}, X \in \mathbb{I}). \tag{3.45}$$

When an inclusion function is used in global optimisation it is important that the bounds on $f(x)$ get better as the interval, X^k, of interest becomes smaller. This property is called the 'convergence order' of the inclusion function.

The order, $\alpha > 0$, of an inclusion, $F(V)$, of $f(V)$, is defined by,

$$w\left(F(V)\right) - w\left(\hat{f}(V)\right) \leq \beta w(V)^{\alpha}, \quad V \in \mathbb{I} \tag{3.46}$$

where β is a positive constant. Thus for small intervals the higher order inclusion functions will produce tighter bounds. However for wide intervals the converse is true and a lower order inclusion will produce tighter bounds. In practice if $w(V) \leq 2$ a second order inclusion will be better than a first order inclusion [36].

The quality of inclusion function is often critical in optimisation applications and so it is important to choose a function inclusion form which is suitable for the problem. The choice of inclusion function directly affects the 'tightness' of the bounds that can be generated. The better the bounds are the fewer subdivisions need to be made and, consequently, the algorithms performance is improved [4]. In general the lower order inclusion functions are better at the start of an algorithm when the intervals, X^k are large and the higher order inclusions are more suitable for small intervals.

In order to construct a class of these inclusion functions it must be assumed that some standard functions are already known such that others may be defined recursively. In many cases the standard inclusion function can be derived, from simple information about $f(x)$.

---

[4] See also (3.61)

Take, for example, a function, $s : \mathbb{R} \to \mathbb{R}$, which is monotonically increasing on $\mathcal{D}$, such as e^x on $\mathbb{R}$. Inclusion functions $S : \mathbb{I} \to \mathbb{I}$ of s and EXP: $\mathbb{I} \to \mathbb{I}$ are easily defined for $X = [\underline{x}, \overline{x}]$

$$\begin{aligned} S(X) &= [s(\underline{x}), s(\overline{x})] \\ \mathrm{EXP}(X) &= [e^{\underline{x}}, e^{\overline{x}}] \end{aligned} \tag{3.47}$$

S is an inclusion of s on $\mathcal{D}$ and $\mathrm{EXP}(X)$ is an inclusion of e^x on $\mathbb{R}$.

Once the base functions have been defined (e.g. $\mathrm{EXP}(X)$) inclusion functions may be constructed using natural extension, one of the centred forms or a combination of these and the rules of Interval Arithmetic (3.42).

3.3.2.1.1 Natural Interval Extension The Natural Extension is the basic inclusion. Inclusions for component functions, $f_i(x)$ are assumed to be known and they are 'linked' together using interval arithmetic. Natural Interval Extension constructs $F(X)$ by replacing x with the appropriate interval X and each component function, $f_i(x)$, with an inclusion $F_i(X)$.

Some of the differences between Real Analysis and Interval Analysis are important for the construction of inclusion functions by Natural Extension. Foremost amongst these is that subtraction and division are not the inverses of addition and multiplication, as is the case with real arithmetic i.e. $A - A \neq 0$ and $A/A \neq 1$. For example,

$$\begin{aligned} [0,1] &- [0,1] = [-1,1] \\ [1,2] &\div [1,2] = [\tfrac{1}{2}, 2]. \end{aligned} \tag{3.48}$$

Thus, the order of evaluation of $A + B - C$ is significant and may affect the quality of the bound produced. Also, the distributive law of Real Analysis holds for certain cases only. For example, $AB + AC$ is typically not as good an inclusion as $A(B + C)$. and A^2 is as good as, or better than, $A \times A$.

3.3.2.1.2 Mean Value Inclusion Functions Mean Value inclusion functions are derived from the Mean Value theorem of real analysis [37]. For $f(x) \in C^1$ and $F'(X)$ an inclusion of $f'(x)$

$$T(X,c) = f(c) + (X-c)^T F'(X), \quad c \in \mathbb{R}^n, c \in X. \tag{3.49}$$

The Mean Value form of an inclusion is of order two [36]. Therefore it will provide tighter bounds on intervals with a small width.

The constant c can be chosen as the midpoint of X and this is often the case as it simplifies the calculation of $X - c$.

A result due to Baumann [38] provides formulae for c such that $T(X)$ gives an optimal upper or lower bound.

A Centre, $c^- \in \mathbb{R}^n$, of a box, $X \in \mathbb{I}^n$, can be defined in terms of the gradient, $F_i'(X) = [u_i, v_i]$, and the endpoints of the box, $[\underline{x_i}, \overline{x_i}]$

$$c_i^- = \begin{cases} \overline{x_i} & v_i \leq 0 \\ \underline{x_i} & u_i \geq 0 \\ (v_i \underline{x_i} - u_i \overline{x_i})/(v_i - u_i) & u_i < 0 < v_i \end{cases} \tag{3.50}$$

such that

$$\mathrm{lb}(T(X, c^-)) \geq \mathrm{lb}(T(X,c)) \qquad \forall c \in X. \tag{3.51}$$

A similar formulation provides c^+ which gives optimal upper bounds.

3.3.2.1.3 Taylor Form Inclusion Function The Taylor Form inclusion is also a centred form. For $f(x) \in \mathrm{C}^2$, given $F''(X)$ an inclusion for the Hessian matrix, $f''(x)$.

$$T_2(X) = f(c) + (X-c)^T f'(c) + \frac{1}{2}(X-c)^T F''(X)(X-c), \tag{3.52}$$

T_2 is an inclusion of $f(x)$. $F''(X)$ may be obtained by automatic differentiation [39]. For functions with a bounded Hessian, the Taylor Form inclusion function is of order two [36].

3.3.2.2 The Interval Newton Method

The Interval Newton method can be used to locate the zeros of a set of simultaneous nonlinear equations. As with the real Newton's Method this can be used to solve global optimisation problems by applying the Interval Newton method to the optimality conditions.

This section will describe how the interval Newton method can be used to find the root of a nonlinear equation, $f : \mathbb{R} \to \mathbb{R}$ which has continuous first derivatives. The generalisation to systems of equations is provided along with the Interval Newton method for optimisation.

Newton's method for solving $f(x) = 0$ uses a tangent, $f'(x)$, to the curve at a point x^k to obtain a step $\Delta x = x^k - x^{k+1}$ by

$$f'(x^k)\Delta x = f(x^k) \tag{3.53}$$

Rearranging provides the familiar form of Newton's method for $x \in \mathbb{R}$

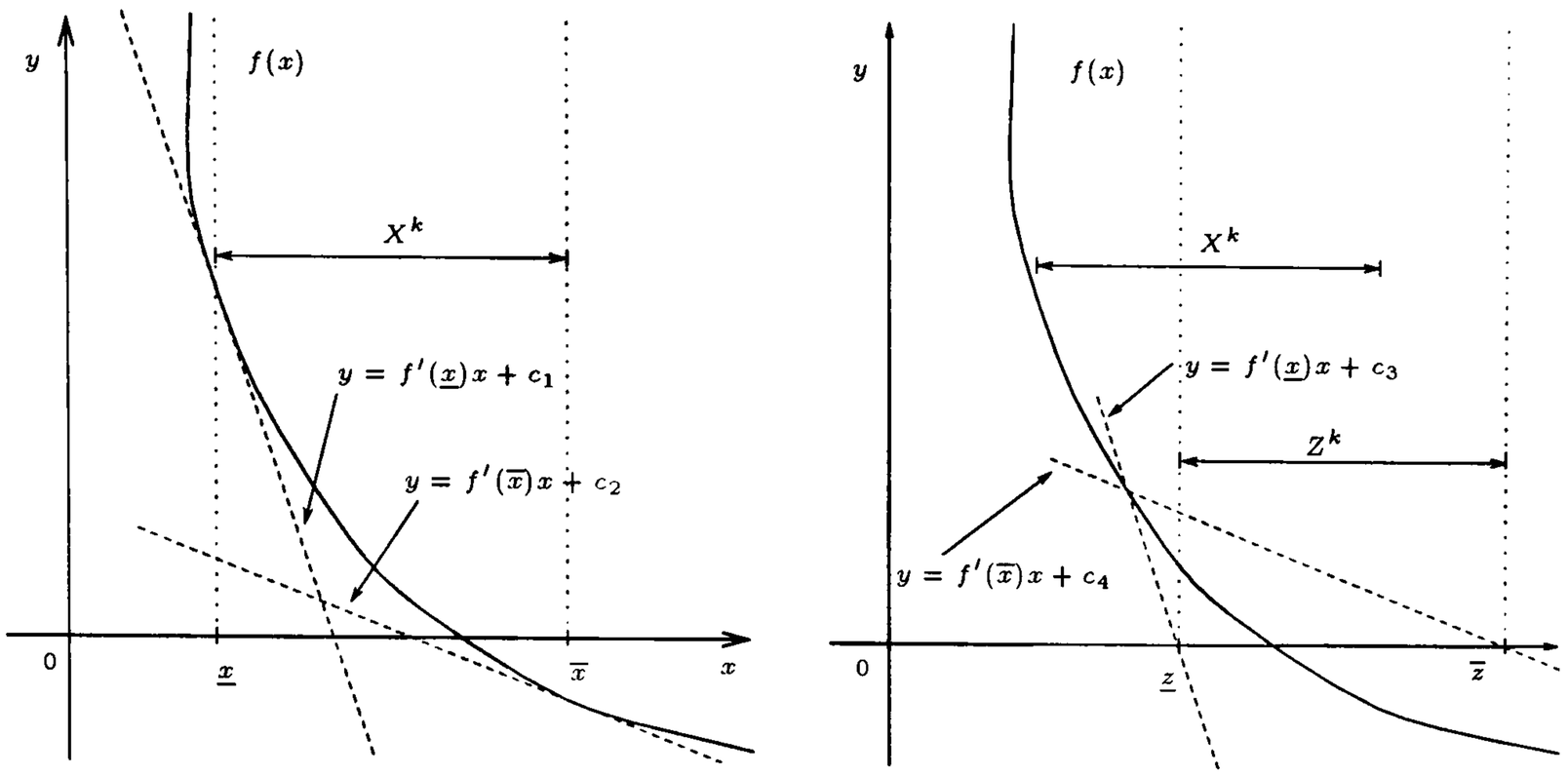
$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}. \tag{3.54}$$

An analogous treatment can be made using interval analysis such that

$$F'(X^k)(Z^k - X^k) = F(X^k) \tag{3.55}$$

$$Z^k = x^k - \frac{F(X^k)}{F'(X^k)} \qquad x^k \in X^k \tag{3.56}$$

where Z^k is the interval given by the next step (analogous to x^{k+1}).



(a) Maximum and minimum gradients of $f(x)$ on X^k

(b) Construction of Z^k for the Interval Newton Step

Figure 3.10: Geometric interpretation of the Interval Newton Algorithm

A geometrical interpretation of the interval Newton step is to construct two lines intersecting at (x^k, y^k). These two lines have gradients equal to the minimum and maximum gradient of $f(x)$ on X^k (see Figure 3.10(a)). As shown in Figure 3.10(b) these two lines form a triangle with the x-axis. The base of this triangle is Z^k and, if X^k contains a root of $f(x)$ then Z^k does also. Assuming that X^k does contain a root of $f(x)$ then so does $X^{k+1} = X^k \cap Z^k$.

If it is assumed that $0 \notin F'(X^k)$ then it follows that for $0 \notin F(X^0)$, X^k will be empty for sufficiently large k. Otherwise the interval will converge on the root with quadratic rate. [40]

To use this algorithm to locate minima, as opposed to roots, of $f(x)$ is simply a matter of locating the roots of $f'(x)$

$$Z^k = x^k - \frac{F'(X^k)}{F''(X^k)} \qquad x^k \in X^k. \tag{3.57}$$

Extending this approach to $\mathbb{R}^n$ requires, as with the real valued Newton method, the solution of a linear system. However, in this case the linear system has interval

coefficients

$$F''(X^k)(Z^k - X^k) = F'(X^k) \tag{3.58}$$

The solution of an interval linear equation is not necessarily compact and is 'generally so complicated in shape that it is impractical to use it' [40]. However, it is possible to obtain an outer approximation to the solution and, in fact, it is common to say that the 'solution' to the problem is an interval containing the solution set. Such a solution might be obtained by applying Gaussian elimination using interval arithmetic in place of real arithmetic.

Gaussian elimination can result in a division by zero and tends to cause large rounding errors. A better alternative is to be obtained by preconditioning $F''(x^k)$ with an approximate inverse of its midpoint and applying the interval Gauss-Seidel method [41].

The Hansen-Greenberg [41] realisation utilises an interval Newton step to accelerate convergence. The method uses just one sweep of the interval Gauss-Seidel procedure to obtain an approximate interval Newton step.

3.3.2.3 Upper Bounds

The interval algorithm is aimed (typically) at locating all the global minimisers of a given problem. For this reason the upper bound is frequently taken to be that given by the interval inclusion. Importantly, this upper bound is an upper bound on the value of the objective (over X^k) which is also an upper bound on the minimum. An upper bound on the minimum is needed by the exclusion phase to remove regions which have been bounded out of the search, this can be obtained by sampling a feasible point.

3.3.2.4 Lower Bounds

The interval inclusion of the objective function provides a lower bound on the value of $f(x)$ over a box, X^k, which is also a lower bound on the global minimum in X^k. For a set of partitions the lowest lower bound is a valid lower bound on NCP.

3.3.2.5 Partitioning

Interval algorithms partition in a number of ways. Partitions are always made by splitting the range of a box. A common method is to bisect a box perpendicular to the longest edge. This technique is simple, guaranteed to reduce the size of the box and consistent with the aim of locating all global minimisers as it provides an even search across the whole feasible region.

Each partition may be reduced in size by the application of the interval Newton step. Then bounds are obtained and the exclusion criteria are checked. If the box can be excluded at this point it will be discarded, if not it is stored. Usually the partition with the longest edge is chosen as the next to partition.

3.3.2.6 Constrained Optimisation

The extension of interval optimisation methods to constrained problems is relatively simple as it uses and builds on the principles used by the unconstrained algorithm. Given the exclusion strategy used in the unconstrained case where regions (or boxes) are excluded according to bounds on the objective function, an extra step can be included to allow exclusion on the basis of infeasibility. Should a box be determined to be infeasible it should be discarded.

3.3.2.6.1 Testing Feasibility Interval analysis can be used to examine the feasibility of a given region with respect to equality and inequality constraints.

For an inequality constraint, $g(x) \leq 0$, with an inclusion function $G(X)$ and an interval $X^k = [\underline{x}, \overline{x}]$. Because interval analysis provides bounds on the value of the constraint function the 'status' of X^k may be determined to be feasible, infeasible or indeterminate [5] subject to $g(x) \leq 0$.

$$\begin{array}{lllll} \text{Feasible} & G(X^k) & \leq & 0, & (\overline{x} \leq 0) \\ \text{Infeasible} & G(X^k) & > & 0, & (\underline{x} > 0) \\ \text{Indeterminate} & G(X^k) & \ni & 0, & \end{array} \tag{3.59}$$

If an interval is indeterminate with respect to a constraint it may contain only feasible points or both feasible and infeasible points.

A box, X, may be determined to contain feasible points with respect to a general equality, $h(x)$, iff $0 \in H(X)$, the inclusion of $h(x)$. Clearly it is not possible for the process of finite subdivision to result in a box which is entirely feasible with respect to an equality, $h(x) = 0$. For this reason some form of relaxation must be made with respect to equality constraints, it is possible to relax the equality to two inequalities by choosing a relaxation constant α.

$$h(x) = 0 \quad \text{is relaxed to} \quad \begin{array}{lll} h(x) & \leq & +\alpha \\ h(x) & \geq & -\alpha. \end{array} \tag{3.60}$$

Alternatively, a maximum box width, β, is chosen and boxes satisfying $w(X) \leq \beta$ are considered to be feasible. This results in a 'chain' of boxes, of acceptable size, along the equality. As with the treatment of inequality constraints, this retains all feasible points.

3.3.2.6.2 Exclusion of Infeasible Regions Infeasibility may be added to the general algorithm as an additional exclusion criteria. However, it will not in general be possible to exclude regions such that the union of the remaining sets, $\mathcal{Z}_j$, is equal

[5] Relational expressions on $\mathbb{I}$ are outlined in §3.3.2, pg 50.

to the feasible region, $\mathcal{A}$, within a finite number of divisions. This is because the division of intervals is orthogonal whereas the constraints, typically, will not be.

Further uncertainty is introduced because the bounds produced by the inclusion functions are not necessarily 'tight'. This uncertainty is reduced as the width of X is reduced. If it holds that,

$$\begin{aligned} & w(G) \rightarrow 0 \\ & w(H) \rightarrow 0 \\ \text{as} \quad & w(X) \rightarrow 0 \end{aligned} \tag{3.61}$$

then the region $\mathcal{Z} = \bigcup_j \mathcal{Z}_j$ can be reduced such that its complement with $\mathcal{A}$ is smaller than a specified tolerance. It is always the case that $\mathcal{A} \subseteq \mathcal{Z}$ so that no feasible points are discarded.

As boxes are divided and bounds on the objective function are accumulated they may be discarded if they do not contain any feasible points or if the lower bound on $f(x)$ over the box is greater than the current estimate of the global optimum, $\overline{y^*}$.

For unconstrained problems obtaining an upper estimate on the value of $f(x^*)$ is simply a matter of sampling any point, $x^k \in X^0$, on the graph of $f(x)$. To obtain this value when the problem contains constraints x^k must be a feasible point. Thus only those x^k that are feasible contribute. This can be improved if a feasible point algorithm is used to determine a feasible point in each indeterminate box. In the case of equality constrained problems it is necessary to use a feasible point search in order to generate values for $\overline{y^*}$.

3.3.2.6.3 Additional Techniques for Constrained Optimisation A number of additional procedures have been included in interval optimisation to enable the solution of constrained problems. Compared to the volume of work which exists for unconstrained optimisation using interval analysis, there is very little experience with constrained problems and, in particular, equality constrained problems [42].

These techniques aim to reduce the search space by using the constraints to create and, subsequently, remove more boxes.

In particular, the interval Newton method can be used to verify the feasibility of a box, X^k, allowing more boxes to be discarded. Application of a Newton step to the constraint set, $h(x) = 0$, provides a smaller box which contains all the feasible points of the larger box or indicates that the box is infeasible [40]. The interval Newton method can also be applied to an interval Fritz-John system, with suitable preconditioning, to reduce the size of a box though this system may become singular [42].

Local optimisation may be used to determine local optimisers so that they can be excluded from the search. The aim is to construct boxes about local minimisers which are small enough that can be discarded if they are bounded out of the search but also for the boxes to be of a size where it is worthwhile discarding them. A technique for constructing a 'well-sized' box about a local optimum is given by Kearfott [42]. Two separate lists of boxes are maintained, the first, as before, holds all the boxes to be searched the second stores boxes which have been determined to contain local minimisers. The list of local minimisers which could not be excluded from the search is then subtracted from the main search list.

Further techniques have been proposed by Vaidyanathan & El-Halwagi [43] including a 'distrust region' approach which uses local optimisation to expand a box around an *infeasible* point so that it can be excluded.

3.4 Summary

The rigorous global optimisation methods presented here are quite varied in their original presentation but they can all be interpreted in terms of the basic covering algorithm which locates global minimisers by branch and bound.

All of the methods presented guarantee global optimality so the decision about which to use depends on the applicability and efficiency of the algorithms.

3.4.1 Efficiency and Applicability of Bounding Methods

One of the most important aspects of rigorous global optimisation is to balance the tightness of the bounds with the cost of obtaining them. For example, an exact bounding procedure such as the generation of the convex envelope of the problem can provide the solution in a single iteration but the generation of the envelope is often more difficult than the optimisation problem. At the other extreme $-\infty$ will always be a valid underestimator of $f(x)$ and can be obtained very 'efficiently' but is of no practical use to a global optimisation algorithm. Effective bounding methods lie between these two extreme cases.

Most covering methods are constructed so that they can take advantage of additional information, such as derivative information, if it is available. However, the properties of the problems which can be solved by different methods vary considerably. Thus, a second compromise must be made as the tighter bounding methods will, in general, be applicable to a smaller class of problems.

An important aspect of applicability is the amount of transformation required. Convex optimisation presents an extreme example: For every nonconvex objective function, NCP, there is a convex objective function with the same solution formulated using the convex envelope of $f(x)$. Thus, all convex optimisation methods are also global optimisation methods and can, *in principle*, be applied to any nonconvex objective for which there exists a convex envelope. Clearly there is a catch here and that is that the construction of a convex envelope can be more difficult than solving the original optimisation problem.

Furthermore, the difficulty of solving the transformed problem must be considered. Any optimisation problem can, in principle, be transformed into an equivalent separable problem [25] by the addition of new variables and constraints but the resulting increase in dimensionality/complexity makes the problem harder to solve. Given that the transformed problem may need to be solved many times this can have a significant impact on the performance of the overall algorithm.

Thus, another compromise must be made between methods that are efficient for a

specific class of problem and can be applied to many problems with the appropriate transformation and those that can be applied to a more general class of problems but are, perhaps, less efficient.

Interval analysis is the most widely applicable of the bounding methods and has the advantage that rounding errors are accounted for automatically in the interval arithmetic. We will consider the interval methods in more detail in preparation for the remaining chapters of this thesis.

3.4.2 Interval Analysis Methods

Interval analysis lies at the 'cheap and loose' end of bounding methods, getting bounds on a function requires just over twice as many calculations as a function evaluation itself. It is looser than the more costly methods which solve a relaxed problem because interval arithmetic deals with the upper and lower bounds only, discarding information about the behaviour of the function between the bounds.

Interval analysis is also one of the most general approaches to bounding. The function to be bounded does not need to differentiable or even continuous for bounds to be generated. This is not typically the case for the relaxed problem methods; if the relaxed problem is not smooth then it is not sure to be solved globally to provide the lower bound.

Research in interval analysis optimisation has exploited some of these factors. The simplicity of interval analysis makes it a good candidate for use in parallel algorithms [44] and the generality has been used, in combination with parallel techniques for solving nonsmooth problems [45]. A considerable body of the literature has dealt with making use of extra assumptions about the structure and properties of the problem. In particular smoothness assumptions allow the use of local optimisation methods for getting better upper bounds to speed up the removal of suboptimal boxes and differentiability leads to interval Newton methods to shrink boxes based on gradient bounding information.

The primary weakness of the interval algorithms is in the solution of highly constrained problems.

3.4.2.1 Constrained Optimisation

Interval methods have been developed primarily to solve bound constrained problems. Results for constrained problems are uncommon and do not show great promise. This section will identify some of the weaknesses and causes for this drawback of the interval methods.

The interval algorithm for constrained optimisation adds infeasibility to the exclusion criteria. This maintains the generality of the interval approach and the algorithm can still be applied to nonsmooth problems to locate all global minimisers [45]. However, most of the research in constrained optimisation has concentrated on solving smooth, differentiable problems and this has aimed to improve the way in which infeasible boxes can be rejected and feasible ones can be reduced.

Without the additional techniques which can be employed under differentiability assumptions this approach works when the minimiser does not lie on a constraint but can be very slow if the solution lies on a constraint, especially if the constraint cannot be bounded tightly [46].

Section 4.1.3 will demonstrate that the performance can be improved under these conditions by using a convex optimisation algorithm to locate feasible points in a depth-first approach using interval analysis as the bounding procedure.

The ability to solve problems with active constraints is important in inequality constrained optimisation but it is vital if equality constrained problems are to be solved.

There has been very little reported computational experience with solving equality constrained problems using interval techniques [42]. While it is very difficult to compare the time taken by different algorithms because of varied implementation

environments the results for solving equality constrained problems are not encouraging. In particular a number of the test problems solved by Kearfott [42] which contain equality constraints had to be dropped from his results because memory requirements were excessive.

3.4.2.1.1 Exploiting Problem Structure Methods such as that proposed by Falk & Soland can immediately be used to solve constrained problems where the constraints are convex. Convex constraints can simply be added to the relaxed subproblem. Thus the class of problems which can be solved with the algorithm for unconstrained optimisation includes those with convex inequalities and linear equality constraints by changing the local solver. The same is true of other approaches which use a local solver to solve the relaxed problem but for the Lipschitz and interval methods the algorithms must be extended to deal with any type of constraints[6].

An important difference between the interval and convex underestimator approaches in the literature seems to be that the convex underestimating algorithms depend on the problem being supplied in analytical form[7]. This allows identification of certain features which can be exploited or reformulated. For example, the αBB approach (§3.2.2.3) is improved by separating linear, concave and bilinear terms [28] and using the best bounding functions for these terms. This identification/reformulation step takes place outside the main iteration loop and so the time taken is amortized over the total number of iterations.

As rigorous global optimisation is impossible with black-box models there are few circumstances in which the problem is presented in some form other than the symbolic (paper) form. One reason for the approach taken in the interval methods is ease of use; identification of convex terms requires additional knowledge from the user. This advantage is effectively diminished by the use of symbolic tools which can be used to perform the reformulation 'behind the scenes' [47].

---

[6] other than simple bounds

[7] the problem is 'delivered' on a piece of paper, so to speak

This leads to the idea that it is possible to improve the solution of constrained problems using interval methods by taking advantage of special terms or structure in the problem. Section 4.2.1 reinterprets the interval bounding operation as a relaxed LP with bound constraints. This allows the inclusion of convex constraints in the formulation. Section 4.2.1 develops some of the possible extensions to interval/relaxation approaches using reformulation techniques.

The next chapter will address the issues of using interval analysis in global optimisation algorithms to improve the solution of constrained problems in particular. Appropriate branching rules are considered first, followed by the development of a depth first interval algorithm which can be used to locate a single guaranteed global minimiser much more efficiently than standard interval approaches. We then consider the application of interval techniques when symbolic information is available, reformulating the interval bounding operation as a bound constrained LP. This allows interval analysis to be used for the nonconvex parts of the problem only, rather than applying it to the whole problem, reducing the number of branch variables and tightening the bounding operation for problems with convex constraints. From this point some tighter linear (affine) relaxations are developed using interval techniques to augment the interval bounding procedure. The chapter concludes with the development of some algorithms based on the results and comparison of these algorithms using some standard test problems.

Chapter 4

Improved Interval Optimisation

Interval analysis provides the most general method of solving global optimisation problems. However, current approaches do not fare well with highly constrained problems, in particular equality constrained problems. This chapter aims to develop algorithmic approaches based on interval analysis which are effective for solving constrained optimisation problems (and possibly better for unconstrained problems).

It has already been noted that interval approaches to global optimisation contained in the literature have concentrated on locating *all* the global minimisers of a given problem. This leads to what is, effectively, a breadth-first branch and bound scheme. The following section outlines why depth-first search is a better scheme when only a single minimiser is sought. It will also consider appropriate partitioning rules and termination criteria for constrained optimisation.

4.1 Branch and Bound in Interval Methods

Consider the approach taken by the Moore-Skelboe algorithm [35]. At each step the box with the widest edge is chosen and bisected perpendicular to the widest edge. This leads to a tendency to choose the boxes nearly in the order that they are generated and to search each box evenly.

An algorithm using interval analysis which locates a single global minimiser should be able to perform a less even, more directed search of the feasible region resulting in fewer branches and and fewer bounding operations. As with other interval algorithms no global minimisers will be discarded. The difference is what properties the algorithm guarantees for the remaining boxes.

4.1.1 Partitioning a Box

Partitioning of boxes depends on two decisions; direction of partition and point at which the partition will be made. The Moore-Skelboe algorithm bisects each box perpendicular to the longest edge. This prevents the production of long thin boxes but can also result in a very even search of each box. A more directed search is obtained by using a measure $F_j(X), j = 1 \ldots n$, called the *smear* [42] where:

$$F_j(X) = F(x_1, \ldots, X_j, \ldots, x_n). \tag{4.1}$$

The edge to be partitioned has the maximum smear.

That is, $F_j(X)$ is the inclusion of $f(x)$ over X, with all but the jth component of X reduced to a point. This results in n extra evaluations of $F(X)$ per iteration but reduces the overall number of partitions that need to be made. The points x_i are usually the point at which the partition is to be made.

An asymmetric test problem was constructed to demonstrate how the alternative bisection approach given by (4.1) is better suited to solving these types of problems.

$$f(x) = \frac{x_1^2}{4} + 4\cos x_1 + 8x_2^2 + x_3^2. \tag{4.2}$$

The problem (4.2) exhibits two global minima.

Given an initial box, $X^0 = [(-5, 8), (-2, 3), (-2, 3)]^T$, and $\epsilon = 10^{-6}$, this problem converges to the same solution for both bisection methods but requires 110 bisections

using (4.1) as compared to 174 with the 'widest-edge' bisection mechanism[1].

$$X^* = \begin{bmatrix} \pm 2.7859 \\ -3 \times 10^{-5}, 2 \times 10^{-5} \\ -1.83 \times 10^{-4}, 1.22 \times 10^{-5} \end{bmatrix}. \tag{4.3}$$

Both these solutions have an objective function value of -1.809312.

The more directed search was also employed by Kearfott [42] to solve equality constrained optimisation problems in an algorithm which used interval Newton steps. However, if the Newton step is not used this bisection criterion is not appropriate for constrained problems.

Consider the following (contrived) example:

$$\begin{aligned} &\min_x && x_1 \\ &\text{s.t.} && \\ &&& x_1 = h(x_2). \end{aligned} \tag{4.4}$$

The bisection criterion is based on the range of the objective function only and, consequently, there will be no reduction of boxes in the x_2 direction. Without a Newton step the algorithm cannot satisfy any of the convergence criteria based on box width. Even with the Newton step, depending on the properties of $h(x_2)$, convergence may be hindered by choosing such partitioning criteria. This may, in part, explain the poor performance of the Kearfott algorithm when applied to constrained problems [49].

Thus, the alternative bisection criterion given in (4.1) is better for unconstrained problems but is not generally applicable to constrained global optimisation.

[1]These results were first presented in [48, 46]

4.1.2 Location of a Single Minimiser

Previous approaches to global optimisation can be split into two groups. Algorithms which solve relaxed problems using convex optimisation aim to locate one minimiser whereas interval approaches aim to find a set of boxes which contain all the minimisers.

We will start by considering how the search for a single minimiser affects the termination criteria because the termination criteria, once established, drives the development of the rest of the algorithm.

4.1.2.1 Termination Criteria

The choice of termination criteria is critically affected by this single minimiser approach. When locating all the minimisers there are two main termination criteria.

In algorithms which choose to partition the widest box the algorithms terminate when the widest box is below a specified width. This leads to an efficient approach which results in a list of boxes which are small enough. However, it doesn't ensure that the range of the objective function over the boxes fulfills any criteria. That is, it is possible to be sure that all the points contained in a box are sufficiently close to a minimiser but it is not possible to say that all the points in the box are epsilon-global minimisers or even that each box contains one such point.

In order to ensure that the boxes only contain epsilon-global minimisers the upper bound of the solution boxes must be within epsilon of the lower bound. This requires that the upper and lower bound of all the boxes must be calculated which means that the entire list must be traversed at *each* iteration [36]. Experience with this, more rigorous, termination criterion indicates that it imposes a considerable penalty in terms of speed [50] and, unlike the box-width based criteria, there is no guarantee that the algorithm will terminate after a finite number of bisections.

The approach proposed by Vaidyanathan and El-Halwagi [43, 51] is a hybrid of the two termination criteria which terminates when the range of the objective over

the widest box is less than ϵ. Assuming that the widest box will also have the widest range of objective function this provides the rigour of the latter criterion with the speed of the former but this assumption does not, in general, hold and the algorithm may terminate when the bounds on the objective over the widest box are considerably tighter than the bounds on some other, thinner, box contained in the list.

Searching for a single minimiser circumvents these problems allowing efficient termination criteria which provide the appropriate bounds on the objective function at the solution.

Because the algorithm aims to locate a single minimiser a depth first search is possible. This depth-first approach chooses the box with the lowest lower bound at each iteration. Thus, the list of boxes to be considered is ordered according to lower bound and the box with the lowest lower bound is chosen for examination at each step. Given a lower bound, $\underline{y}$, on $f(x)$, a point $x^k \in X^k$ is an epsilon global minimiser if

$$f(x^k) - \underline{y} \leq \epsilon. \tag{4.5}$$

Thus, the point x^k is definitely a minimiser as defined by (1.3) and there is no need to obtain bounds on all the possible boxes at each iteration.

4.1.3 A Depth-first Interval Algorithm

Given the termination criterion in (4.5) an interval algorithm for locating a single minimiser can be constructed.

At each iteration, k, the box with the lowest lower bound should be chosen from the list of possible boxes. The termination criteria are checked. If the termination criteria are not satisfied the box is split and for each box a lower bound, $\underline{y}$ on $f(x^*)$ can be obtained by interval analysis from $F(X)$ and an upper bound from $f(x^k)$

where x^k is a feasible point in X^k. The new boxes are added to the list and the procedure is repeated.

In practice this means that the points x^k must also be stored with the possible boxes so that the termination criteria can be checked. The cost of this is small and it has other advantages if the feasible point is to be determined by a local optimisation algorithm. If a feasible point in each box is stored then this point will be in one of the two boxes produced by the partition and the algorithm can reuse this point whilst determining a new one for the second box.

Because the depth-first search chooses the box with the lowest lower bound there is no possibility of examining one of the boxes which has been bounded out of the search as these will be at the end of the list. Therefore the algorithm will operate in the same way even if these boxes are not excluded. Experience with this algorithm, even on small problems, indicates that the exclusion step shortens the list and improves the overall performance whilst reducing the memory requirements so this step is included.

The overall algorithm is as follows:

1. Initialise
   (a) a counter, $k = 0$
   (b) an initial region, $X_0 \supseteq \mathcal{A}$.
   (c) store a triple $\mathcal{X}_0 = \{X = X_0, \underline{y}_0 = lb(F(X)), x_0\}$ where x_0 is not set.
   (d) an upper bound on the global minimum, $\overline{y} = \infty$.
2. Select and remove a stored triple, $\mathcal{X}_k$, with the lowest stored value of $\underline{y}_k$.
3. If x_k is not set then set x_k equal to a local minimiser or a feasible point of $\mathcal{A} \cap X_k$ if one can be determined by the local minimisation phase.
4. If $f(x_k) - \underline{y}_k \leq \epsilon$ then
   (a) x_k is a global minimiser and satisfies Eqn. 1.3.

(b) Terminate.

5. Set $\overline{y} = \min\{f(x_k), \overline{y}\}$.

6. Partition $\mathcal{X}_k$ giving $\mathcal{X}_L$ and $\mathcal{X}_R$.

7. Update $\underline{y}_L$ and $\underline{y}_R$. Store $\mathcal{X}_L$ and $\mathcal{X}_R$.

8. Remove any stored triples, $\mathcal{X}_j$ $j = 0 \ldots k$ for which:

   (a) X_j is completely infeasible

   (b) $\underline{y}_j > \overline{y}$

9. Increment k and return to step 2.

4.1.3.1 Application

This algorithm has been applied to the 'Six Hump Camel Back Function' from [36],

$$f(x_1, x_2) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \tag{4.6}$$

with $X^0 = [(-10, 5), (-7, 4)]$. The problem has 15 stationary points in the region of interest. The two global minima are at $[0.0898, -0.7126]^T$ and $[-0.0898, 0.7126]^T$. The results are summarised in Table 4.1, the breadth-first algorithm terminates when the list contains only epsilon-global minimisers using the standard breadth-first search.

Table 4.1: Result summary for 4.6.

Algorithm	Iterations	Time to solve (s)	Time per iteration ($s \times 10^3$)
Width first	374	53	141.3
Depth first	200	6	30

The two algorithms have also been applied to a constrained test problem which is given in (4.15). The width-first algorithm does not converge to a solution because

Table 4.2: Problem (4.15) using depth first search.

Iterations	Time to solve (s)	Time / iteration ($s \times 10^3$)
567	30	52

the feasibility criteria are not satisfied. The depth-first approach converges to a single epsilon global minimiser in 30 seconds. These results are given in table 4.2.

This section has described an interval inclusion algorithm using local optimisation in a depth-first manner to improve the solution of constrained and unconstrained problems and provide, on termination, a single guaranteed epsilon-global minimiser.

This improvement in interval optimisation has been made without any extra assumptions about the problem, its properties or representation other than those required by the local optimisation step. In the next section the process of obtaining interval bounds is recast as a linear programming problem which will allow interval algorithms to exploit structure and representation of NCP.

4.2 Exploiting Symbolic Information and Problem Structure

In §3.4 it was noted that the different global optimisation approaches make a variety of assumptions about the properties of NCP and the information, such as derivatives, which is available. This means that some approaches are more efficient but applicable to fewer problems. This section is motivated by the principle that

the best algorithm is the one which uses all the available information.

Current approaches to global optimisation which use interval analysis as the bounding technique seem to assume that the problem is defined by a model, similar in

concept to a black-box model, which provides the necessary bounds and/or derivative information but does not provide a symbolic representation.

This means that interval methods can be applied to problems which fulfill this assumption as well as to problems expressed in a symbolic form. It does not, however, exploit the presence of such a symbolic representation.

Given that rigorous global optimisation is impossible with real black-box models it is very unusual for global optimisation problems to be expressed in a non-symbolic form. This is adequately demonstrated by the paucity of such problems in the interval literature, in fact all of the common test problems are expressed in symbolic form.

The results presented in the following sections are independent of the branching scheme and the termination criteria used in the algorithm. This thesis examines the application in a depth-first fashion but could, equally, have used the breadth-first approach.

4.2.1 Reformulation of NCP

This section considers how reformulation of problems given completely or partially in symbolic form can be used to improve the application of interval algorithms. Three important reformulation techniques will be considered. Two of them are quite similar so they will be called the α and β substitutions, the third is a smoothing reformulation which can be used with certain max/min constructions.

4.2.1.1 The α–substitution

The α-substitution transforms, by addition of a new variable α, the problem so that the objective function becomes a constraint. The following optimisation problem

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & \\ & g(x) \leq 0 \end{aligned} \tag{4.7}$$

is equivalent to

$$\begin{aligned} \min_{\alpha,x} \quad & \alpha \\ \text{s.t.} \quad & \\ & g(x) \leq 0 \\ & f(x) \leq \alpha \end{aligned} \tag{4.8}$$

This substitution is most useful in combination with the smoothing reformulation but by itself it has two advantages. The objective function is now linear in α and the objective and constraints are expressed in the same form. Thus, all or part of the objective function can be turned into constraints and, for convenience, we need only consider constraints for the following reformulations.

4.2.1.2 The β–substitution

The β-substitution adds a new variable to split a constraint into two simpler constraints. The single inequality given by

$$g(u(x)) \leq 0 \tag{4.9}$$

becomes two constraints

$$g(\beta) \leq 0 \quad \text{and} \quad \beta = u(x). \tag{4.10}$$

Suppose that one of $g(x)$ and $u(x)$ is nonlinear then $g(u(x))$ is also, the transformed problem now has one nonlinear constraint and one linear constraint. The important advantage is that the nonlinearity has been isolated in a single constraint. This substitution can be applied recursively to reduce a complex nonlinear constraint to a set of simpler constraints.

4.2.1.3 The Smoothing Reformulation

This is not so much a 'reformulation' as a property of optimisation problems which allows a reformulation. This property is vitally important to global optimisation as it allows a constraint such as

$$\max_j\{u_j(x)\} \leq 0 \qquad j = 1 \ldots m \tag{4.11}$$

which is not usually differentiable everywhere due to the max operator, to be written as m smooth/differentiable constraints

$$u_j(x) \leq 0 \qquad j = 1 \ldots m. \tag{4.12}$$

This transformation is possible because only the tightest of the constraints in (4.11) can be active at any given time. Consequently, this technique does not work for

$$\begin{array}{rcl} \min_j\{u_j(x)\} & \leq & 0 \\ \text{or} & & \qquad j = 1 \ldots m \\ \max_j\{u_j(x)\} & \geq & 0 \end{array} \tag{4.13}$$

One situation in which the transformation can be applied is when, for example, there are two ways of relaxing a given constraint. Consider $g(x) \leq 0$ where $g(x) \geq u_1(x)$ and $g(x) \geq u_2(x)$ then

$$\begin{array}{rcl} g(x) & \geq & u_1(x) \\ g(x) & \geq & u_2(x) \end{array} \tag{4.14}$$

so, if two or more underestimators are known for $g(x)$ they can *both* be used in a relaxed problem.

4.2.2 Relaxed Problems from Reformulation

The reformulations described in §4.2.1 can be applied to any optimisation problem which is given in symbolic form. For many of the test problems in the interval literature the transformations make little or no difference because the problems have few variables and the objective and constraints are usually composed of entirely nonconvex terms. Again the test problems do not reflect the kind of problems that occur in practice which typically are composed of a number of terms many of which will be convex or linear. This is particularly the case in larger problems where nonconvexity can often be isolated to a subset of the constraints. Of the 21 chemical engineering applications given by Ryoo & Sahinidis [52] 19 have at least one convex constraint and 15 have at least one linear constraint. All the problems have some linear terms in the constraints or objective function.

4.2.2.1 Relaxing Nonconvex Objective Functions

As noted in §3.4 global optimisation methods which solve a convex relaxed problem to obtain bounds can incorporate convex inequalities and linear equalities in the relaxed problem without any modification. An analogous approach can be taken using interval analysis by reformulating the problem.

This is best illustrated with an example. Consider the following problem from [18],

$$\begin{aligned} \min_{x} \quad & -x_1 + x_1 x_2 - x_2 \\ \text{s.t.} \quad & \\ & -6x_1 + 8x_2 \leq 3 \\ & 3x_1 - x_2 \leq 3 \end{aligned} \tag{4.15}$$

where $x_1 \geq 0$ and $x_2 \leq 5$.

The problem has two minimisers at $[0.916, 1.062]$ and $[1.167, 0.5]$ with $f(x) = -1.0052$ and -1.0833 respectively. The global minimiser lies on a constraint.

This problem has one nonconvex term, x_1x_2, but the rest are linear. Using the α-substitution it can be transformed to

$$\begin{aligned} \min_{x,\alpha} \quad & -x_1 + \alpha - x_2 \\ \text{s.t.} \quad & \\ \alpha - x_1x_2 &= 0 \\ -6x_1 + 8x_2 &\leq 3 \\ 3x_1 - x_2 &\leq 3 \end{aligned} \tag{4.16}$$

and using interval analysis the first constraint can be relaxed to $\alpha \in X_1X_2$ providing an underestimating LP,

$$\begin{aligned} \min_{x,\alpha} \quad & -x_1 + \alpha - x_2 \\ \text{s.t.} \quad & \\ -6x_1 + 8x_2 &\leq 3 \\ 3x_1 - x_2 &\leq 3 \\ x &\in X \\ \alpha &\in X_1X_2. \end{aligned} \tag{4.17}$$

The bounds obtained by solving this LP will be tighter than the usual interval bounds because the solution is always feasible with respect to the original linear constraints. This doesn't necessarily mean that the algorithm will be faster as the cost of solving the LP is higher than that of evaluating the interval bounds.

Recall from §3.3.2.1 that the Mean Value theorem of interval analysis can be used to obtain tighter bounds than the natural extension for boxes below a certain width. Ratschek & Rockne [36] proposed that the natural extension could be used for boxes with a width greater than 2 and the mean value form for the smaller boxes. The

smoothing reformulation allows both sets of interval bounds to be used so there is no need to decide which will be tighter as the reformulation will ensure that the tightest bound is active.

Another advantage is that the number of variables which need to be branched on is reduced. Without the reformulation all the variables need to be partitioned and branched. The reformulated problem, when solved with an LP/NLP, only branches on the variables which occur in nonlinear/nonconvex terms respectively. For the problem given by (4.15) there is no reduction but for problems where a small number of variables appear in nonconvex expressions, for example in the Haverly Pooling problem (5.1), this can significantly improve the search.

This reformulation of the interval bounding approach as a linear programming problem is achieved with one assumption; all or part of the global optimisation problem is presented in a symbolic form. The approach can still be applied to any problem to which interval analysis may be applied but the advantage is that convex constraints are satisfied by the lower bound.

This is especially advantageous if there are linear equality constraints as these will make the bounds from the reformulated interval problem considerably tighter than those of the interval extension of the objective function.

4.2.2.2 Relaxing Nonconvex Constraints

A similar approach can be taken when nonconvex terms appear in the problem constraints. Many problems have a linear objective with some convex and some

nonconvex constraints. Consider the following problem from [17];

$$\begin{aligned} \min_{x} \quad & -x_1 - x_2 \\ \text{s.t.} \quad & \\ x_1 x_2 &\leq 4 \\ x_1 - 4x_2 &\leq 0 \\ 0 \leq x_1 &\leq 4 \\ 0 \leq x_2 &\leq 8 \end{aligned} \tag{4.18}$$

The global solution is -5 with $x^* = (4, 1)$.

The nonconvexity arises from the bilinear term in the first constraint but the objective function is linear, as is the second constraint. This problem can be relaxed using interval analysis to provide a lower bound on $x_1 x_2$. Suppose $X_1 X_2 = [\underline{\alpha}, \overline{\alpha}]$ by Natural Extension and that the Mean Value form of $X_1 X_2$ is $[\underline{\beta}, \overline{\beta}]$ then

$$\begin{aligned} \min_{x \in X} \quad & -x_1 - x_2 \\ \text{s.t.} \quad & \\ \underline{\alpha} &\leq 4 \\ \underline{\beta} &\leq 4 \\ x_1 - 4x_2 &\leq 0 \end{aligned} \tag{4.19}$$

is a suitable linear relaxation of (4.18) for a given interval, X. And provides a tighter lower bound than the interval extension of the objective function on X. If this problem is infeasible then X does not contain any feasible solutions.

The advantage of this approach is more fully realised when there are few nonconvex constraints involving a subset of the variables.

This section has recast the problem of obtaining lower bounds with interval analysis as a linear/convex optimisation problem. It is possible to replace the objective and each constraint with an interval extension to obtain the same lower bound as the interval approach. However, identification of convex terms in the objective and

constraints allows a subset of the terms to be relaxed using interval analysis. This means that the number of variables which need to be branched on is restricted to those that appear in nonconvex terms and that the lower bound produced is feasible with respect to any convex constraints.

This particular relaxation using interval analysis to relax nonconvex terms in the objective or constraints will be called the 'Interval Bound' relaxation or I.B relaxation.

All of these improvements have been made by assuming that all or part of the optimisation problem is provided in a form such that the linear/convex components can be identified. Further improvements can be made with extra assumptions allowing the construction of tighter, linear, bounds from interval analysis.

4.2.2.3 Linear Interval Bounds from Natural Extension

Now that the problem of getting lower bounds has been recast as an optimisation problem it is possible to use both the Mean Value and Natural Extension forms to provide bounds on the nonconvex terms. Bearing this in mind this section develops some more bounding techniques which can be applied to the same problems and can be added to this optimisation formulation to improve it.

Consider the bilinear term to be bounded in (4.15), $\alpha = x_1 x_2$. At each iteration bounds, X_1 and X_2, are known for x_1, x_2. So far this constraint becomes

$$\begin{aligned} \alpha &\in X_1 X_2 \\ &\in [\min(\underline{x_1}\underline{x_2}, \underline{x_1}\overline{x_2}, \overline{x_1}\underline{x_2}, \overline{x_1 x_2}), \max(\underline{x_1}\underline{x_2}, \underline{x_1}\overline{x_2}, \overline{x_1}\underline{x_2}, \overline{x_1 x_2})] \end{aligned} \tag{4.20}$$

by the Natural Extension of $x_1 x_2$. The lower bound from interval analysis is more conservative than need be and it is possible to leave, say, x_1, as a continuous variable,

$$\alpha \geq \min(x_1 \underline{x_2}, x_1 \overline{x_2}). \tag{4.21}$$

This min term, however, is not smooth for all X_1 and is not of the right form for the smoothing reformulation (§4.2.1.3) to be applied. It is not smooth because the left hand term is smaller when x_1 is positive and larger when x_1 is negative so that the function is not differentiable at $x_1 = 0$. This means that it is not possible, in general, to solve a linear program containing this term to global optimality.

Fortunately, for this example, $x_1 \geq 0$ which means that the problem can be solved. More generally, as the interval X_1 will be split during the application of the algorithm it is sufficient to ensure that $x_1 = 0$ only ever occurs at the edge of X_1. This can be achieved by splitting each interval X_i at zero in the first iteration.

Again it is possible to add as many relaxations of this constraint as necessary so it is also possible to use the Natural Extension and, this time, keep x_2 as a continuous variable. This gives a relaxed form of (4.15),

$$\begin{aligned}
&\min_{x,\alpha} \quad -x_1 + \alpha - x_2 \\
&\text{s.t.} \\
&\qquad \alpha \geq \min(x_1\underline{x_2}, x_1\overline{x_2}) \\
&\qquad \alpha \geq \min(x_2\underline{x_1}, x_2\overline{x_1}) \\
&\qquad \alpha \in X_1X_2 \\
&-6x_1 + 8x_2 \leq 3 \\
&\qquad 3x_1 - x_2 \leq 3
\end{aligned} \tag{4.22}$$

This approach can also be used to tighten the interval relaxation of other terms by keeping some variables or parts of variables continuous. For example, $x^2 = xx$ so that

$$\begin{aligned}
x^2 &\geq \min(x\underline{x}, x\overline{x}) \\
x^3 &\geq \min(x\underline{x}^2, x\overline{x}^2) \\
x^n &\geq \min(x\underline{x}^{n-1}, x\overline{x}^{n-1})
\end{aligned} \tag{4.23}$$

which are smooth on $[\underline{x}, 0]$ and $[0, \overline{x}]$. Upper bounds can be obtained in a similar fashion.

These linear bounding functions obtained by using continuous variables in the Natural Interval Extension will be referred to as the NE underestimators or NE relaxation.

The NE underestimators can be used as an extended form of interval arithmetic to provide additional linear bounding information in interval calculations. The rules for this arithmetic are briefly described in §B.2.

Given that the Mean Value inclusion sometimes provides better bounds than the Natural Interval inclusion it may be useful to take an analogous approach with the Mean Value Extension to derive MV underestimators based on the Mean Value inclusion.

4.2.2.4 Linear forms of the Mean Value Inclusion

The Mean Value inclusion for $f(x)$ given by (3.49),

$$T(X,c) = f(c) + (X-c)^T F'(X), \quad c \in \mathbb{R}^n, c \in X. \tag{4.24}$$

provides tighter bounds than the Natural Extension inclusion over smaller boxes if the gradient of $f(x)$ is available. It is possible to apply the techniques described above to introduce continuous variables into the Mean Value form.

Consider,

$$f(x) \in f(c) + (x-c)^T F'(X), \quad c \in X. \tag{4.25}$$

Say $F'(X) = [\underline{d}, \overline{d}]$ then, taking the same approach to multiplication as with the NE form, the lower bound is given by

$$f(x) \geq f(c) + \min\{(x-c)^T \underline{d}, (x-c)^T \overline{d}\}. \tag{4.26}$$

This linear underestimator is not smooth at c and so, cannot be included in the relaxed problem if $\underline{x} < c < \overline{x}$. Given that c changes it is not possible to ensure

that the first split of the box X^0 will place c on the edge of X^k, as with the NE underestimators. It is possible, however, to restrict c to the edge of a given interval X, so that the sign of $x-c$ is constant over X, providing two smooth underestimators

$$\begin{aligned} f(x) &\geq f(\underline{x}) + (x - \underline{x})^T \underline{d} \\ f(x) &\geq f(\overline{x}) + (x - \overline{x})^T \overline{d} \end{aligned} \tag{4.27}$$

and two overestimators

$$\begin{aligned} f(x) &\leq f(\underline{x}) + (x - \underline{x})^T \overline{d} \\ f(x) &\leq f(\overline{x}) + (x - \overline{x})^T \underline{d}. \end{aligned} \tag{4.28}$$

Applying this to the bilinear term in (4.15) with c at the vertices of $X_1 \times X_2$ gives

$$\begin{aligned} \alpha &\geq \underline{x_1}\,\underline{x_2} + (x_1 - \underline{x_1})\underline{x_2} + (x_2 - \underline{x_2})\underline{x_1} \\ \alpha &\geq \overline{x_1}\,\overline{x_2} + (x_1 - \overline{x_1})\overline{x_2} + (x_2 - \overline{x_2})\overline{x_1} \end{aligned} \tag{4.29}$$

as relaxations for $\alpha \geq x_1 x_2$.

These MV linear underestimators are applicable to any differentiable term. For example, $\alpha \geq -x^2$ is relaxed to

$$\begin{aligned} \alpha &\geq -\underline{x}^2 + (x - \underline{x})(-2\overline{x}) \\ \alpha &\geq -\overline{x}^2 + (x - \overline{x})(-2\underline{x}). \end{aligned} \tag{4.30}$$

The last two sections have introduced two new types of relaxation based on linear versions of the Natural Extension and the Mean Value inclusion. The following section demonstrates how these relaxations, along with the interval inclusions, can be used to construct a lower bounding LP.

4.2.3 Linear Relaxations of NCP

The work on exploiting symbolic information is very general and can be applied to improve interval algorithms when some parts of the problem can be identified as convex. This section will consider a specific variant of those ideas to construct linear lower bounding problems as we will use these later.

The implementations of the ideas from the previous sections construct a linear underestimating problem. That is, even terms which are known to be nonlinear but convex are relaxed. The reason for testing the ideas in this restricted formulation will become clear later when the algorithm is applied to the solution of modular flowsheet optimisation. In summary, a linear underestimator can be easily characterised and the LP can be solved without re-evaluating the flowsheet model.

4.2.3.1 MV Relaxation

Given a constraint $g(x) + c^T x \leq 0$ where $g(x)$ is nonlinear, interval arithmetic gives $g(X) \in [\underline{g}, \overline{g}]$ and the constraint can be relaxed to $\underline{g} + c^T x \leq 0$ which is linear.

If $g(x)$ is differentiable and $g'(x) \in [\underline{d}, \overline{d}]$, by interval extension of the gradient of $g(x)$, then the MV relaxation gives

$$\begin{aligned} \beta &\leq -c^T x \\ \beta &\geq g(\underline{x}) + (x - \underline{x})^T \underline{d} \\ \beta &\geq g(\overline{x}) + (x - \overline{x})^T \overline{d} \\ \beta &\leq g(\underline{x}) + (x - \underline{x})^T \overline{d} \\ \beta &\leq g(\overline{x}) + (x - \overline{x})^T \underline{d}. \end{aligned} \tag{4.31}$$

which are, again, linear in x.

Although it is not strictly necessary to introduce additional variables (β) into the constraints for each group of nonlinear terms it makes it easier to formulate the

relaxed problem automatically. The constraints

$$\begin{aligned} \beta &\leq -c^T x \\ \beta &\geq g(\underline{x}) + (x - \underline{x})^T \underline{d} \end{aligned} \tag{4.32}$$

and

$$g(\underline{x}) + (x - \underline{x})^T \underline{d} \leq -c^T x \tag{4.33}$$

are equivalent. Introduction of extra variables also simplifies the use of interval inclusion because the interval inclusion of $g(x)$ can be used in the bound constraints of β.

Figure 4.1(a) shows the underestimators for $x^2 + 10$ on $[-2, 3]$ constructed with the MV relaxation. The MV underestimators are given by,

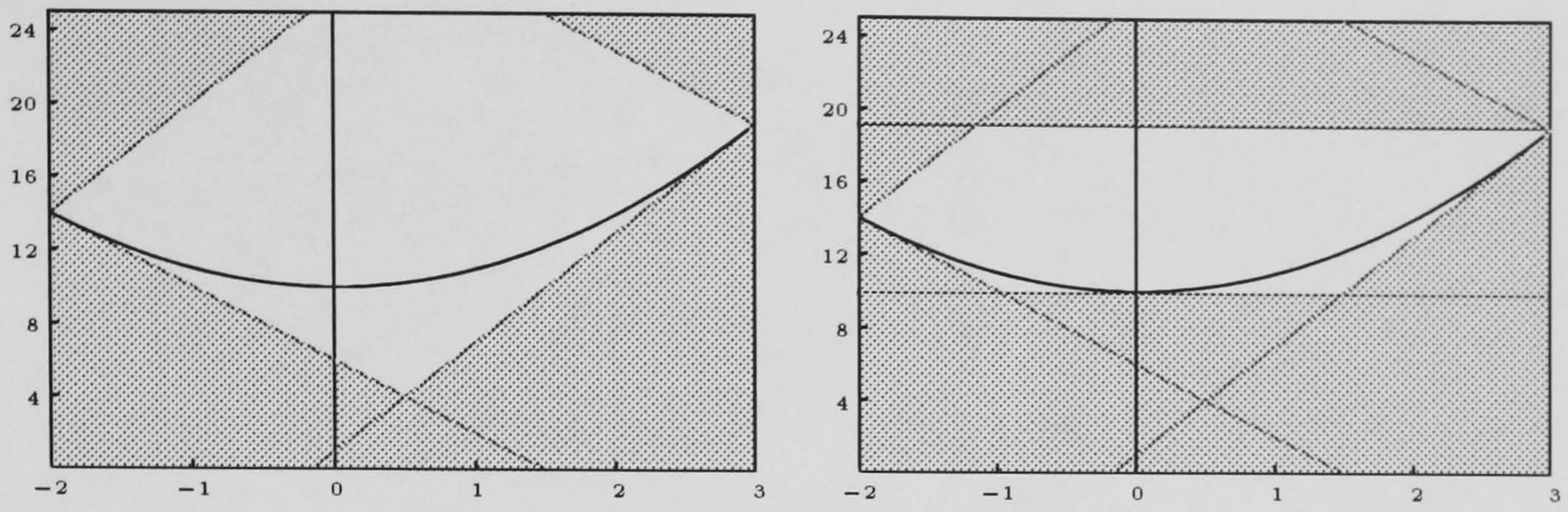
$$\begin{aligned} x^2 + 10 &\geq \overline{x}^2 + 10 + (x - \overline{x})2\overline{x} \\ x^2 + 10 &\geq 19 + (x - 3)6 \end{aligned} \tag{4.34}$$

and

$$\begin{aligned} x^2 + 10 &\geq \underline{x}^2 + 10 + (x - \underline{x})2\underline{x} \\ x^2 + 10 &\geq 14 + (x + 2)(-4). \end{aligned}$$

Using interval inclusion of $x^2 + 10$ provides the extra relaxations shown in Figure 4.1(b). It can be seen that the underestimators for $x^2 + 10$ are much tighter than the overestimators which are comparatively loose. Using the interval bounds as well provides a much tighter relaxation overall. The figure shows that, for convex functions of a single variable, the upper bounds from interval inclusion can make the upper bounding function considerably tighter than the MV relaxation alone (for concave terms the same is true for the lower bound) and that using both MV and interval inclusion is tighter than either method alone.

The relaxations are obtained by constructing MV over and underestimators at each of $\underline{x} = (\underline{x_1}, \underline{x_2} \ldots \underline{x_n})$ and $\overline{x} = (\overline{x_1}, \overline{x_2} \ldots \overline{x_n})$. Further relaxations can be obtained by



(a) MV Relaxation

(b) MV Relaxation and Interval Inclusion

Figure 4.1: Relaxations of $x^2 + 10$

constructing MV underestimators at other corners of X. In particular, the relaxation for a bilinear term $\alpha = x_1 x_2$ is

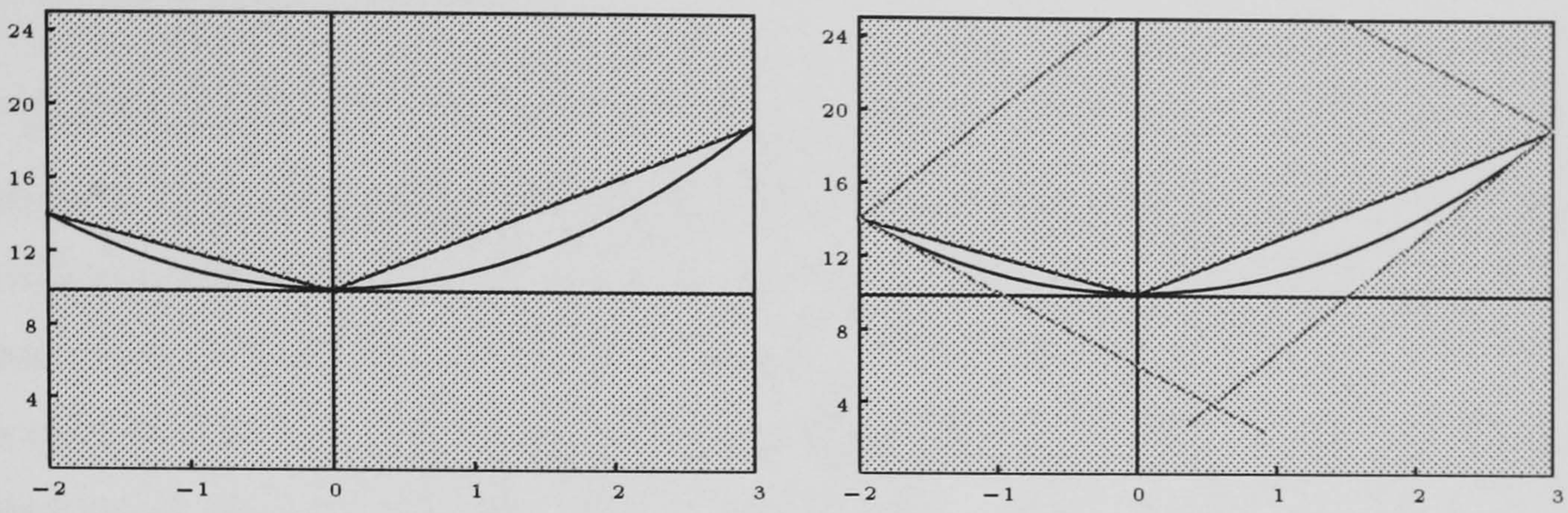
$$\begin{aligned} \alpha &\geq \underline{x_1}\,\underline{x_2} + (x_1 - \underline{x_1})\underline{x_2} + (x_2 - \underline{x_2})\underline{x_1} \\ \alpha &\geq \overline{x_1}\,\overline{x_2} + (x_1 - \overline{x_1})\overline{x_2} + (x_2 - \overline{x_2})\overline{x_1} \end{aligned} \tag{4.35}$$

$$\begin{aligned} \alpha &\leq \overline{x_1}\,\underline{x_2} + (x_1 - \overline{x_1})\underline{x_2} + (x_2 - \underline{x_2})\overline{x_1} \\ \alpha &\leq \underline{x_1}\,\overline{x_2} + (x_1 - \underline{x_1})\overline{x_2} + (x_2 - \overline{x_2})\underline{x_1} \end{aligned}$$

The relaxations at the other corners can be added but they will be redundant.

Experience with these relaxations indicates that a naive application, which uses *all* the constraints, does not suffer a large penalty in execution time in the LP solver[2] which is important when it is not possible to determine which relaxations are nonredundant.

[2]in this case the Matlab 4.2c LP



(a) NE Relaxation

(b) MV & NE Relaxation

Figure 4.2: NE Relaxations of $x^2 + 10$ on $[-2, 0]$ and $[0, 3]$

4.2.3.2 NE Relaxation

Although the NE relaxation cannot be applied on the interval $[-2, 3]$ it is possible to apply it to either half of the range around the origin. The addition of NE relaxations can further improve the tightness of the relaxation over large intervals (i.e early in the algorithm) and splitting at $x = 0$ in iteration 1 is frequently useful by itself when using MV underestimators.

Figure 4.2 shows how the NE underestimators can improve the tightness of the relaxation when used in combination with the MV underestimators.

The NE relaxation is not as general as the MV relaxation and the interval inclusion but when it can be used it provides a useful addition to the relaxed problem.

The following sections describe two depth-first algorithm which can use different combinations of MV, NE, MV & NE and interval inclusion relaxations of the objective function. Results of application will consider how important the different sets of underestimators are in practice by applying them to a selection of global optimisation test problems.

4.3 Algorithms

The developments of the previous section (§4.2) have been implemented in two depth-first branch and bound algorithms using interval analysis as the primary bounding method and using extra symbolic information where possible. The two variants are called Interval Global Optimisation Relaxation (IGOR) and Reduced Interval Global Optimisation Relaxation (RIGOR) because they use interval analysis and relaxation to efficiently solve constrained global optimisation problems.

Though the ideas are implemented in a depth-first algorithm there is no reason why they should not be applied in the classical interval breadth-first algorithm to locate all the global minimisers.

Interval analysis can be used to bound a very broad range of functions without any differentiability assumptions. The M.V underestimators can be used to bound differentiable terms by reformulation of the interval bounding problem as a LP. The NE underestimators and the reformulations described in §4.2.1 can be used when symbolic information is available.

4.3.1 The IGOR Algorithm

The IGOR algorithm is a depth-first branch and bound algorithm using interval inclusion and MV relaxation to construct a lower bounding LP. At each iteration the lowest box is removed from the list and bisected perpendicular to its longest edge[3]. Lower bounds are obtained for the new boxes and the boxes are returned to the list if the lower bounding problem is feasible.

[3]The edges are scaled according to the widths of X_i^0

4.3.1.1 Lower Bounds

For some box X^k the relaxed LP, P^k is solved to obtain lower bounds. If NCP is of the form

$$\begin{aligned} \min_x \quad & f(x) + c^T x \\ \text{s.t.} \quad & \\ g(x) + a^T x &\leq 0 \\ Ax - b &\leq 0 \end{aligned} \tag{4.36}$$

where $f(x)$ and $g(x)$ are differentiable on X^k, then P^k is

$$\begin{aligned} \min_{x,\alpha,\beta} \quad & \alpha + c^T x \\ \text{s.t.} \quad & \\ \max_i\{f_l(x, c_i)\} - \alpha &\leq 0 \\ \beta + a^T x &\leq 0 \\ \max_i\{g_l(x, c_i)\} - \beta &\leq 0 \\ Ax - b &\leq 0 \end{aligned} \tag{4.37}$$

where $c_i \in \mathbb{R}^n$ are the corners of X^k, $f_l(x, c_i)$ and $g_l(x, c_i)$ are the MV relaxations of $f(x)$ and $g(x)$ at c_i, the max(...) terms are rewritten according the the smoothing formulation in §4.2.1.3 and bounds on the new variables α and β are obtained from the interval inclusions of $f(x)$ and $g(x)$ on X^k respectively. NE relaxations of $f(x)$ and $g(x)$ may also be added.

Note that if the NE and MV relaxations are dropped this becomes the IB relaxation which was defined in §4.2.2.2. If the relaxation is simplified further so that *all* terms in the objective and constraints are bounded then this reduces to the depth-first approach outlined in §4.1.3. The IGOR algorithm is logically a superset of the depth-first interval algorithm.

4.3.1.2 Upper Bounds

The IGOR algorithm gets upper bounds by sampling the solution to the lower bounding LP. If the solution, x^{k*}, is feasible for NCP then the upper bound is updated.

Better upper bounds can be obtained by solving NCP with a local optimisation algorithm using x^{k*} as a starting point. However, in all but one case of application to the test problems, the convergence of the algorithm is limited by the lower bounding problems. In short, use of a local algorithm typically determines the global minimum in the first couple of iterations but the lower bound takes much longer to meet it. The extra overhead of solving NCP at each iteration does not seem to be justified. The one case where convergence is limited by the upper bound is in a concave problem.

4.3.2 The RIGOR Algorithm: Reduction

The RIGOR algorithm adopts the same approach as IGOR but an additional 'Reduce' step is added. Given an upper bound, $\overline{y}$, on the solution to NCP and a lower bounding LP, P^k, which we rewrite here as a LP with the extra variables incorporated into the vector x:

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & \\ & Ax \leq b. \end{aligned} \tag{4.38}$$

The reduction step solves a set of LPs to tighten the bounds on each of the x variables from NCP. For each x_i that appears nonlinearly in NCP solve

$$\begin{aligned} \min_{x \in X^k} \quad & x_i \\ \text{s.t.} \quad & \\ & c^T x \leq \overline{y} \\ & Ax \leq b. \end{aligned} \tag{4.39}$$

to tighten the lower bound, $\underline{x_i}$, and solve

$$\begin{array}{ll} \max\limits_{x \in X^k} & x_i \\ \text{s.t.} & \\ & c^T x \leq \overline{y} \\ & Ax \leq b. \end{array} \tag{4.40}$$

to tighten the upper bound. If these problems are infeasible then the box, X^k can be deleted.

It is possible that tightening the bounds on, say, x_1, will result in a tighter relaxation and, consequently, a better reduction problem for $x_i, i > 1$. This, however, requires construction of a new problem and hence extra evaluations of the gradients. If the gradients can be evaluated and a new problem constructed efficiently then this would be a better option for the reduce step. RIGOR does not use this technique and only one set of reducing LPs is constructed for each box.

A number of other reduction steps are used by the BARON algorithm [52] and in the analytical approach proposed by P. Hansen *et al* [53]. The BARON approach includes additional tests which do not require the solution of a relaxed problem. These tests have not been implemented in RIGOR.

The tightening steps used in RIGOR but not present in IGOR are especially useful when the initial bounds, X^k, are much larger than the feasible region described by the constraints in NCP. For example, the following problem from [54],

$$\begin{array}{ll} \min\limits_{x} & x_3 \\ \text{s.t.} & \\ & x_1 + x_2 - x_4 = 0 \\ & x_1\, x_2 + x_3 = 0 \end{array} \tag{4.41}$$

where X^0 is given by

$$\begin{array}{lll} 0 \leq x_1 \leq 50 & \qquad & -100 \leq x_3 \leq 100 \\ 0 \leq x_2 \leq 50 & & -100 \leq x_4 \leq 100. \end{array}$$

Application of a single reduction step tightens X^0 to $x_1, x_2 \leq 4$. The bounds on the other variables are not tightened because they appear linearly in the problem.

4.3.2.1 Upper Bounds

Because RIGOR uses the best current upper bound, $\overline{y}$, there is more justification for using a local optimisation algorithm to obtain upper bounds. However, given that local optimisation usually determines a good upper bound on the first application (at least with this problem set) RIGOR uses a local NLP algorithm to determine an upper bound, $\overline{y}$, before the first bisection. The NLP uses the center of X^0 as the starting point for the optimisation. For the rest of the iterations RIGOR uses the same method as IGOR.

4.3.3 Results of Application to Test Problems

The IGOR and RIGOR algorithms have been applied with a variety of different relaxations. The test problems are as follows.

camel The six hump camel function from (4.6). This problem has bound constraints only.

gop3 The illustrating example for the GOP algorithm in [17].

gos2 The motivating example for the GOS algorithm in [18] and also given by (4.15).

haverly The Haverly Pooling problem from [55, 52][4] and given by (5.1).

ssb A 'simple bilinear program' from [54] which is used by the author to demonstrate the effectiveness of reduction techniques. Given in (4.41).

[4]Note: The problem is misprinted in [55] but the correct formulation was used by the Authors

ryoo20 Problem 20 from the process design examples given in [52]. This problem is a reactor network design originally presented by Manousiouthakis & Sourlas [56].

and from [57]

fpqc2 Quadratically constrained test problem number 2.

fpqp3 Quadratic program number 3.

fpqp5 Quadratic program number 5.

nlp3 Design of a three stage process with recycle. This problem was originally presented by Stephanopolous and Westerberg [58].

nlp4 Nonlinear programming problem 4.

nlp6 Nonlinear programming problem 6.

The number of variables, constraints and bound constraints are listed in Table 4.3 and all the problem descriptions are provided in Appendix C. The final column indicates which of $f(x), g(x), h(x)$ is nonconvex.

Each problem is attempted with three relaxation techniques

NE The Natural Extension linear underestimators from §4.2.2.3 and Natural Extension interval inclusion.

MV The Mean Value underestimators from §4.2.2.4 and the Natural Extension interval inclusion.

MVNE Both MV underestimators and NE underestimators with Natural Extension interval inclusion.

Table 4.3: Statistics for the Test Problem Set

	Variables	Constraints	Bounds	Nonconvexity	ϵ
camel	2	4	4	f	10^{-3}
fpqc2	5	16	10	f, g	10^{-2}
fpqp3	14	22	13	f	10^{-5}
fpqp5	10	21	10	f	10^{-4}
gop3	2	6	4	g	10^{-6}
gos2	2	4	2	f	10^{-4}
haverly	9	25	18	h, g	10^{-3}
nlp3	4	9	6	f	10^{-4}
nlp4	4	9	6	f	10^{-4}
nlp6	2	6	4	g	10^{-3}
ssb	4	11	8	g	10^{-4}
ryoo20	6	17	12	h, g	10^{-4}

Table 4.4 lists the number of iterations, and CPU time for convergence to the global optimum using IGOR. The two averages are calculated differently. Mean (T) is the mean for the total number of problems solved by each method. Mean (S) is the arithmetic mean of the results for problems which *all* methods solved successfully. Table 4.5 lists the same statistics for the RIGOR algorithm.

All of these results are obtained using Matlab 4.2c on an IBM RS6000[5]. The relaxed LP was solved using the LP solver from the Matlab Optimisation Toolbox and the upper bounding NLP used in RIGOR is solved using the SQP algorithm, also from the Optimisation Toolbox.

Some of the results are fairly predictable; IGOR performs significantly better on the 'camel' problem than RIGOR because this problem is unconstrained, the 'ssb' problem was deliberately constructed with loose bounds and an additional, unnecessary, variable to demonstrate reduction techniques [54] so RIGOR performs substantially better than IGOR. Interestingly, the non-reducing algorithm employed by Smith [54]

[5]Model 140–43p

Table 4.4: Results for IGOR using different relaxations

	NE		MV		MVNE	
	Iter	Time (s)	Iter	Time (s)	Iter	Time (s)
camel			372	41.6	362	44.08
fpqc2	4693	1396.00	44	24.40	44	29.06
fpqp3	193	127.80	19	9.62	8	4.17
fpqp5	133	62.99	55	31.51	55	33.28
gop3	38	2.75	2	0.72	2	0.61
gos2	52	5.45	18	2.28	16	2.00
haverly	86	14.9	3	1.21	3	1.20
nlp3	26	3.37	14	2.34	14	2.43
nlp4	26	3.28	14	2.82	14	2.70
nlp6			233	9.87	167	9.33
ssb	202	16.0	33	3.64	33	3.65
ryoo20			1249	513.00	831	369.00
Mean (T)	133.1	29.6	152.7	54.7	107.5	41.6
Mean (S)		29.6		6.8		6.3

requires investigation of 609 nodes with an NLP solution at each, IGOR requires a total of 33 bisections solving only 66 LP relaxations.

4.3.3.1 MV vs. NE

The NE relaxation performs better than MV in only two of the problems, nlp3 and nlp4, using RIGOR. In both cases the objective function is concave subject to linear constraints. This means that the global minimum must lie at a vertex of the feasible region and, if the NE relaxation is tight at this point then the extra computational effort used in the MV relaxation is because of the increased number of constraints in the reduction phase. Aside from these two results MV relaxation is consistently better than the NE relaxation. The gaps in the table indicate that the NE relaxation when used alone failed to solve some of the problems and exceeded the maximum time allowed.

Table 4.5: Results for RIGOR using different relaxations

	NE		MV		MVNE	
	Iter	Time (s)	Iter	Time (s)	Iter	Time (s)
camel			162	130.7	154	142.77
fpqc2			3	13.92	3	19.8
fpqp3	171	846.0	18	113.6	8	49.0
fpqp5	5	29.8	4	16.96	3	20.93
gop3	12	2.57	2	0.92	2	0.92
gos2	38	7.02	6	2.0	6	1.94
haverly	85	13.94	3	3.1	3	3.15
nlp3	2	0.41	3	1.46	2	0.48
nlp4	2	1.04	3	1.54	2	0.61
nlp6			211	28.39	151	22.86
ssb	32	12.47	4	1.82	4	1.92
ryoo20					26	163.0
Mean (T)	87.7	114.2	38.1	28.5	30.3	35.6
Mean (S)		150		22.5		14.9

4.3.3.2 MV vs. MVNE

The MVNE relaxation is, overall, more efficient than the MV relaxation alone for IGOR but not for RIGOR. This is because the penalty for increasing the number of constraints is much higher for RIGOR because more LP problems are solved per iteration and some of the constraints will be redundant.

4.3.3.3 IGOR vs. RIGOR

Overall RIGOR performs better than IGOR. The number of iterations (nodes examined) is always lower but that is to be expected. The time taken is a better measure of the difference between the two algorithms.

For all the problems except ssb, initial bounds, X^0 were deduced from the constraints of the problem. These bounds may not have been tight but they are not very loose

either. The ssb problem has very loose bounds because of the way it is formulated and RIGOR performs much better than IGOR on this problem. It is to be expected that the initial bounds will frequently be more like those of ssb than the other problems.

RIGOR does not perform all the possible tightening LPs. Only if x_i appears with a positive coefficient in the constraints of (4.40) is problem (4.40) solved. If x_i appears with a negative coefficient then (4.39) is solved. Given the results of applying RIGOR to the camel problem, which is unconstrained, it would seem that tightening of x_i should only occur if x_i appears in the constraints of NCP. This would lead to an algorithm which reduces to IGOR if the problem has only bound constraints and so, would, be more efficient overall for constrained and unconstrained problems.

4.4 Summary

This chapter has developed a number of techniques for improving interval based optimisation. Depth first search allows the location of a single rigorous ϵ-global minimiser. Reformulation of the interval bounding problem as a convex NLP allows the inclusion of convex constraints in the lower bounding phase and reduces the number of branch variables to those that appear in nonconvex terms. The MV and NE underestimators derived from the Mean Value and Natural Extension inclusions provide tighter bounds than the normal interval inclusion under circumstances where extra information is available. The use of MV and NE relaxations also allow the incorporation of the reduction (tightening) steps in the RIGOR algorithm.

Some of the techniques developed here have been used to implement two algorithms, IGOR and RIGOR, which use linear relaxations of NCP to solve constrained global optimisation problems. The two algorithms have been tested with a set of standard nonconvex test problems using different combinations of the relaxation techniques.

The results of application to the test set do not conclusively indicate that one approach is superior but, overall, the use of RIGOR with MV or MVNE relaxation

has been shown to be the best option from those tested.

Though the relaxations are based on bounding nonconvex terms the IGOR and RIGOR approaches bound all nonlinear terms, including convex terms. This is not the most efficient approach for problems where the relaxed problem is cheap to evaluate. In the following chapter the linear relaxation will be exploited in the optimisation of modular flowsheets where the number of flowsheet evaluations must be kept to a minimum.

Chapter 5

Global Optimisation of Modular Systems

Section 4.2.1 considered some of the ways in which symbolic representations of global optimisation problems can be exploited in interval based algorithms to improve speed and reliability for constrained problems. The work in the previous chapter is motivated by the fact that very few global optimisation problems are given in a non-symbolic form. However, a large class of problems, which have not been addressed in the global optimisation literature, does exist where the problem is not provided analytically. These are the problems where modules, whose behaviour is known, are connected together into a system whose overall expression is not available in analytical form.

For Chemical Engineering one important class of such problems is the modular flowsheeting model. In developing a methodology for solving this class of problems globally we will begin with a motivating example.

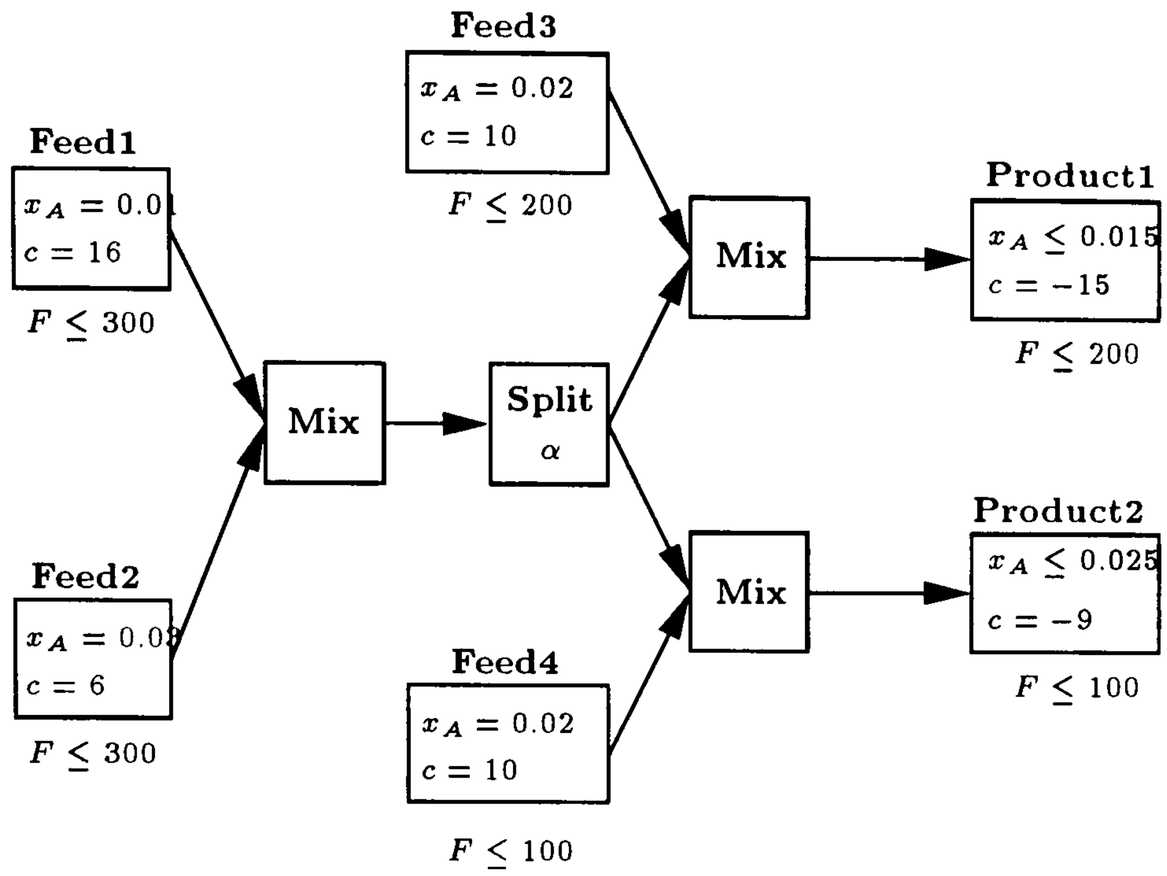


Figure 5.1: The Haverly Pooling Problem

5.1 Motivating Example

Consider the Haverly Pooling problem given in figure 5.1 as formulated by Quesada & Grossmann [59]. The problem is to blend four feeds into two products to minimise the total cost. Each feed has a fixed composition, x_A, and cost, c(£/Kmol hr). Each product has a cost, required composition and flowrate, F. The feeds are mixed to produce products which satisfy the quality requirements using mixer and splitter units to represent the different blending tanks.

The equation based formulation from [55] is

$$\begin{aligned}
\min_{F,x_A} \quad & 6F_1 + 16F_2 + 10F_4 - 9F_5 + 10F_7 - 15F_8 \\
\text{s.t.} \quad & \\
F_1 + F_2 - F_3 - F_6 &= 0 \\
F_3 + F_4 - F_5 &= 0 \\
F_6 + F_7 - F_8 &= 0 \\
0.03F_1 + 0.01F_2 - F_3x_A - F_6x_A &= 0 \\
F_3x_A + 0.02F_4 - 0.025F_5 &\leq 0 \\
F_6x_A + 0.02F_7 - 0.015F_8 &\leq 0
\end{aligned} \tag{5.1}$$

where x_A is the mole fraction of component A after mixing. This is the formulation used in the equation based optimisation with IGOR in §4.3.3.

The problem is modular. All the mixers are the same, as are all the splitters. The feeds are unit operations without inputs where the flowrate is a parameter of the unit, the product units represent quality requirements.

Given that this is one of many pooling/blending problems that may need to be solved it would be advantageous to be able to formulate the problem as a modular problem; placing units onto the flowsheet and 'drawing' the streams to connect them. It is natural to view the problem in terms of interconnected units with each unit performing some transformation of the input stream to provide the output stream.

This allows unit models to be reused in other flowsheets reducing the amount of work required to formulate the problem and reducing the scope for error in the formulation. It presents a view of the flowsheet which is the intuitive view for the Process Engineer.

The problem is nonconvex so a local optimisation algorithm will not guarantee the global solution but Global optimisation algorithms cannot be applied to a normal modular flowsheet.

5.2 Formulation of Modular Flowsheets

There are many ways of formulating flowsheeting problems. Two important distinctions are identified in the literature; at one end of the spectrum lies the Equation Oriented flowsheeting approach and, at the other, the Sequential Modular flowsheet. In Equation Oriented (EO) flowsheets the flowsheet is treated as a set of mass/energy balance equations which are solved simultaneously. The Sequential Modular (SM) approach views the flowsheet as a set of interconnected black-box models of the unit operations. Comprehensive reviews of flowsheeting techniques are given by Biegler [60] and Perkins [61].

Both approaches have their advantages and, consequently, a large body of research has been devoted to constructing flowsheeting methodologies with the best mixture of properties. This is particularly the case for flowsheet optimisation where the sequential component of SM flowsheeting results in poor performance as the flowsheet is converged at every optimisation step. This has lead to Simultaneous Modular flowsheets which move towards the EO end of flowsheeting paradigms by treating recycle convergence as a constraint in the optimisation problem.

An EO flowsheet can be constructed so that it behaves more like a modular flowsheet by using procedures to calculate intermediate values from explicit equations. In the following section the definition of a very general module to describe unit operations allows us to start with the modular flowsheet and gradually migrate towards a more equation oriented approach. This general model will be used in the development of the modular global optimisation approach to present the two conceptual endpoints of the possible modular approaches to global optimisation.

5.2.1 The General Unit Module

To define models for unit operations it is necessary to construct a general model for unit operations. This model is given in figure 5.2. This is a general model which can be applied to any unit operation. We say that models of specific unit operations

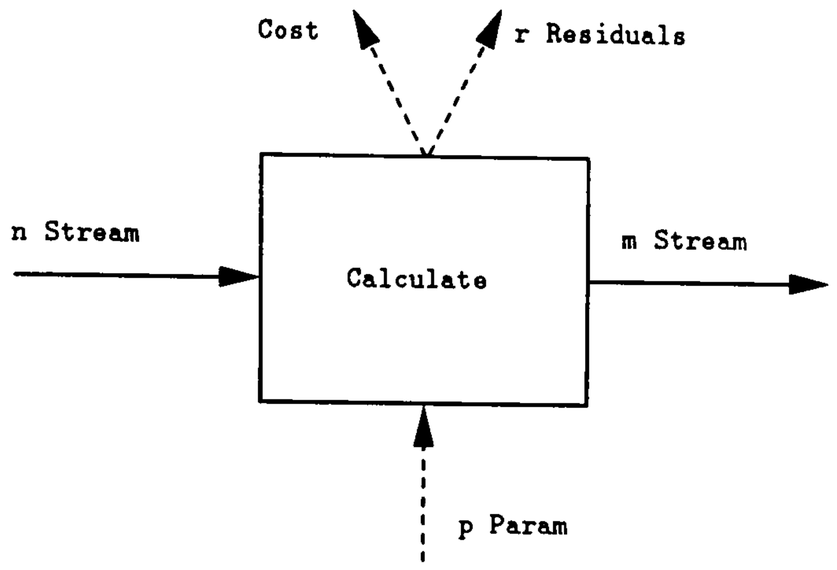


Figure 5.2: A General Unit Module

are 'instances' or 'specialisations' of this general model.

Each unit, j, has n input streams which are the outputs of the preceding unit and m output streams which are the feeds for successive units. There are p input parameters, ϕ, which determine how the unit operates, for example the split fraction in a splitter, and a cost, γ, which represents some cost associated with the unit. There are r residuals, ρ, which represent constraint violations.

The Haverly problem requires feeds, products, splitters and mixers. A feed has no input streams ($n = 0$) and one output stream. The flowrate is determined by a single input parameter and the cost by multiplying the unit cost, c, by the flowrate. A mixer has two inputs and one output calculated by 'adding' the inputs together; in this case the cost is zero. Splitters divide one input into two outputs based on the input parameter and again the cost is zero. The quality constraints placed on the two product streams become residuals in the product modules which have one input stream and no outputs.

The Haverly problem does not have a recycle but a technique for dealing with recycles is vital for a general flowsheeting methodology. To maintain a view consistent with the general model, a stream can be torn by introducing a 'tear unit' which has an input, an output and a residual based on equality of the two streams. The 'tear unit' parameters are the output stream values as shown in figure 5.3. The tear unit is a conceptual tool, similar to a 'control block', which represents tearing a stream

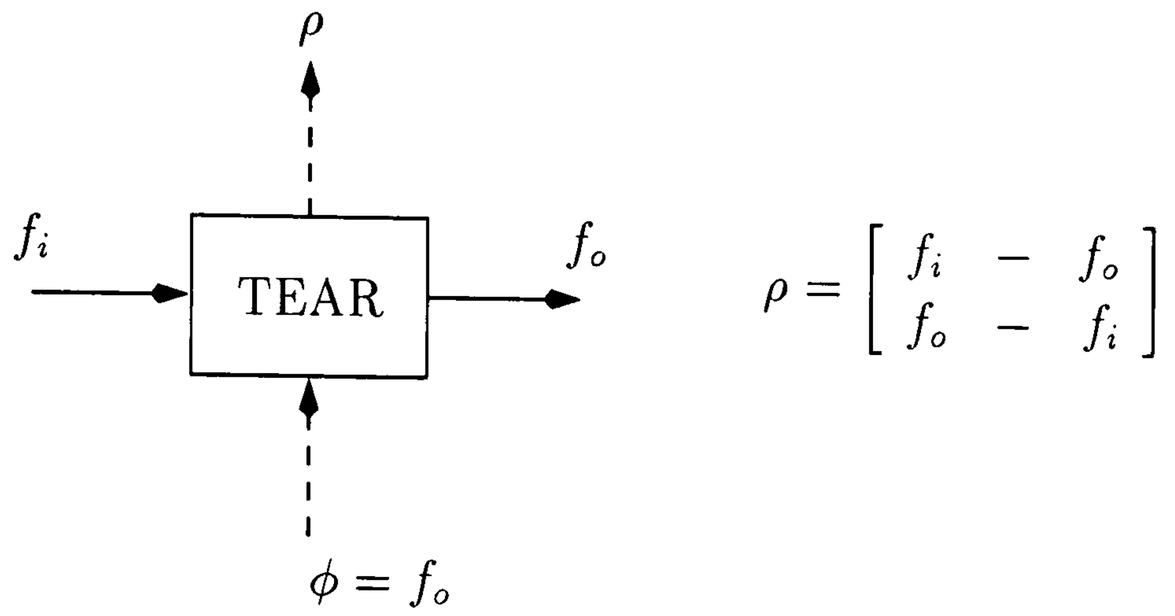


Figure 5.3: A Tear Unit Module

it is not a real unit.

5.2.2 Different Formulations

The optimisation problem is to minimise the sum of the costs subject to the residuals being less than or equal to zero by manipulating the unit parameters. That is,

$$\begin{aligned} \min_{\phi} \quad & \sum_j \gamma_j \\ \text{s.t.} \quad & \\ & \rho \leq 0 \end{aligned} \tag{5.2}$$

There are some distinct options for optimising flowsheets constructed from these general modules

1. Modules are constructed such that only tear and product units have residuals, all other units calculate outputs from operating parameters, and

   (a) the flowsheet model is optimised by *first* solving the tear residuals and then applying an optimisation step

   (b) The optimisation step uses the tear and product residuals as constraints

2. All modules have residuals, the parameters are the output streams and

   (a) Some of the modules are solved and then the remaining residuals are solved as optimisation constraints

   (b) All of the residuals are solved simultaneously as part of the optimisation constraints.

These different constructions start with a SM flowsheet and evolve towards a more equation based approach as calculations are moved from the flowsheet to the optimisation. Option 2a does not describe a unique formulation for a given flowsheet but one such unique formulation is to use calculation in the module where an explicit expression for the outputs is available and residuals where an equation must be solved. This reduces the size of the optimisation problem and means that the explicit modules will always be feasible.

In terms of local solution of the problem (5.2) these general modules are another way of looking at a well studied problem. They will be put in context in the following sections which develop the ideas for a modular approach to global optimisation and determine how closely the global flowsheet should follow the SM flowsheet.

5.3 Extended Type Flowsheets

An extended arithmetic type is an abstraction that makes it easier to perform calculations on arithmetic operators to obtain information about the calculation.

This section describes extended types and the generic models which make it possible to write suitable unit operations. With this basis the rest of the section considers a few of the possible extended types which might be used for modular flowsheet optimisation and introduces some of the features of the optimisation algorithms which can be applied with these 'types'.

5.3.1 Extended Arithmetic Types

An interval is a compound type made from two real numbers which is used to represent a range. An extended type is made from the compound type plus the rules that define operations on it. Thus, the interval compound used according to the rules of interval arithmetic is an extended type which, for convenience, is also called an interval.

This is a specialised case of the more general ideas of Object Oriented Programming (OOP). Objects, or types, are collections of data and functions (operators) which form a useful abstraction.

This thesis has already considered a number of extended arithmetic types

- Intervals are an extended type for calculating the results of application of operators to ranges.
- Vectors and matrices are, in principle, extended types built from arrays of scalar types and rules to manipulate the data.
- The convex underestimators proposed by McCormick [25] and described in §3.2.2.2 provide the rules for adding, subtracting and multiplying convex underestimators. A suitable definition of the compound object which maintains the data required at each step makes this an extended type.
- The automatic differentiation rules proposed by Rall [39] are, when combined with a compound for the data, an extended type. Importantly the AD compound is built with arrays of some underlying type which can be a real number or an interval.
- The linear interval bounds proposed in §B.2 form an extended type. Notice that these linear bounds are a specialisation of the convex bounding functions and a logical superset of the rules of interval arithmetic.

All of the operations provided by these extended types can be obtained with normal real arithmetic. The key to the encapsulation of data and operator rules is that the

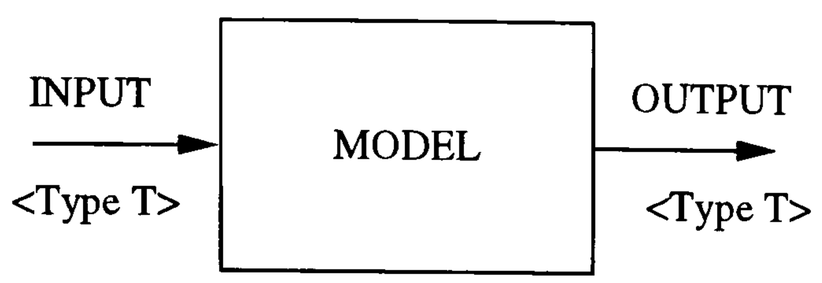


Figure 5.4: A generic model

abstraction provided by the extended type reduces the complexity of the calculations and allows one abstraction to be built upon another. Also the fact that these techniques are very useful in the construction of software cannot be ignored because flowsheeting packages and global optimisation algorithms will ultimately need to be implemented in software.

Rather than construct many different flowsheets from units based on different types some form of model which can deal with types generically is necessary.

5.3.2 Model Genericity

A generic model (as opposed to a 'general' model) is a model that specifies the transformations that need to be applied to some underlying type, T, to obtain the output, as shown in figure 5.4. The model needs to describe the operations and then the appropriate rules are applied for T. For example, a module which adds its inputs should use the interval arithmetic rules if the underlying type is an interval (T = interval) and the rules of AD if the underlying type is an AD type.

This is not an unusual idea, a vector arithmetic is a generic way of applying operations to vectors of some type, T. When two real vectors are added the elements are added componentwise, when two interval vectors are added the intervals are added componentwise. The addition of the components differs but the rule for adding vectors is the same.

Once the types are constructed they can be used with the model to give the appropriate information. Because the generic model specifies *which* operations (or

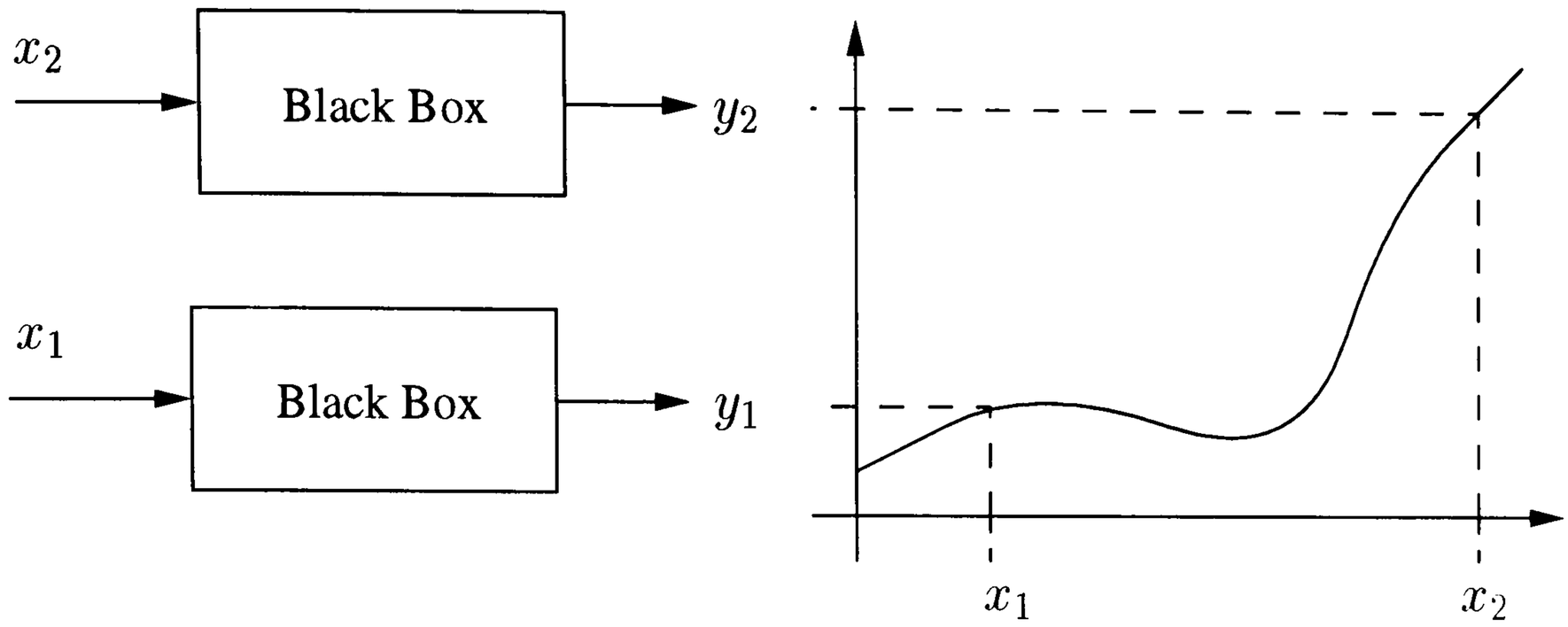


Figure 5.5: Black Box Models

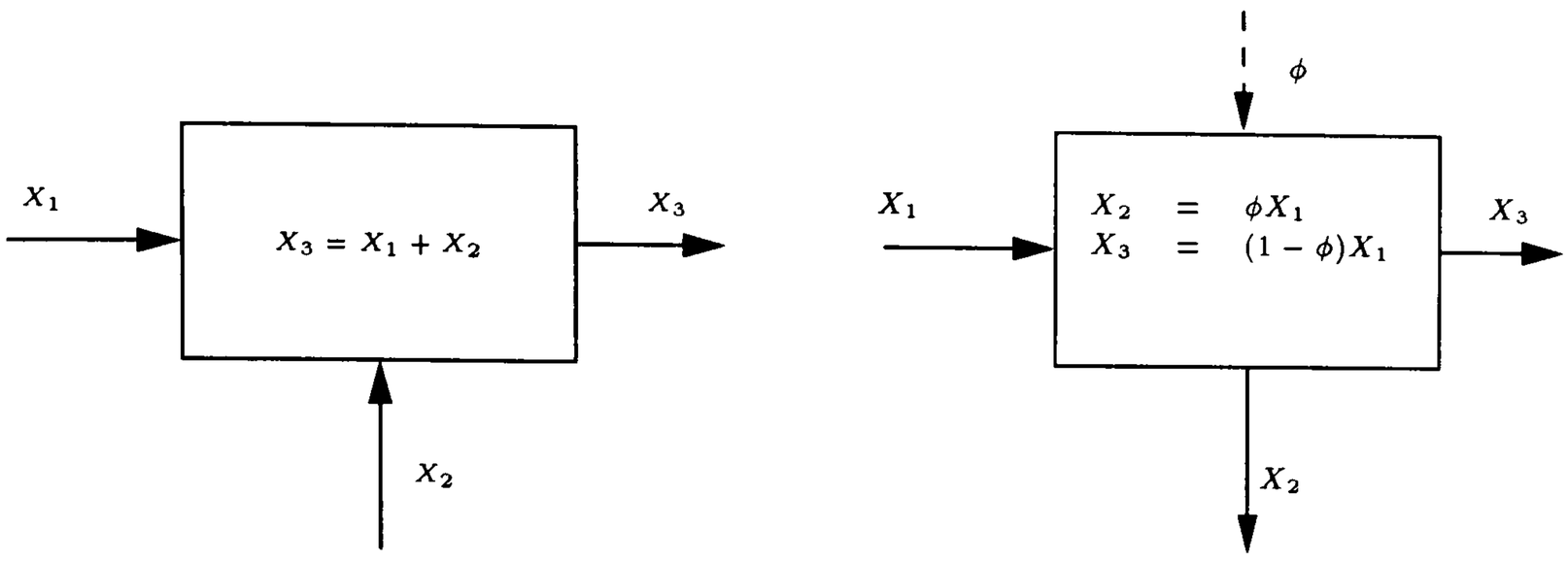
transformations) are to be applied and the type defines *how* those operations are applied to the compound, a generic model can be used with types that are unknown when the model is constructed. The generic mixer is the same as the interval mixer in figure 5.6(a) which is the same as the real model of a mixer.

5.4 Interval Modular Flowsheets

The following sections develop a modular flowsheeting approach which can be globally optimised and algorithms which can be applied to optimise such a flowsheet. The starting point for this modular approach is the Interval Unit Operation.

5.4.1 Interval Unit Operations

The problem with solving a modular flowsheet of the pooling problem globally is that the flowsheet behaves like a black box model because it is built from unit operations which are connected together. The information transferred between units is only about the current point.



(a) An Interval Mixer

(b) An Interval Splitter

Figure 5.6: Unit Operations Based on Interval Arithmetic

In order solve the optimisation problem globally it is necessary to obtain upper and lower bounds on the problem. Figure 5.5 illustrates the problem of obtaining the necessary bounds from black box models. It is only possible to evaluate the output at fixed points which provides no information about the behaviour of the model between those points. It is not possible to minimise the model rigorously if it is nonconvex.

If these models are rewritten using interval arithmetic so that the inputs are intervals and the outputs are intervals which bound the possible real outputs each unit model provides the information necessary to apply an interval global optimisation algorithm to the unit (see figure 5.6(a)). Then, as with a normal modular flowsheet, it is possible to link unit operations with streams, which use intervals instead of real numbers, to get bounds on the outputs of the flowsheet.

This provides a means by which modular flowsheets can be constructed from unit operations based on interval analysis such that a global optimisation algorithm can be applied.

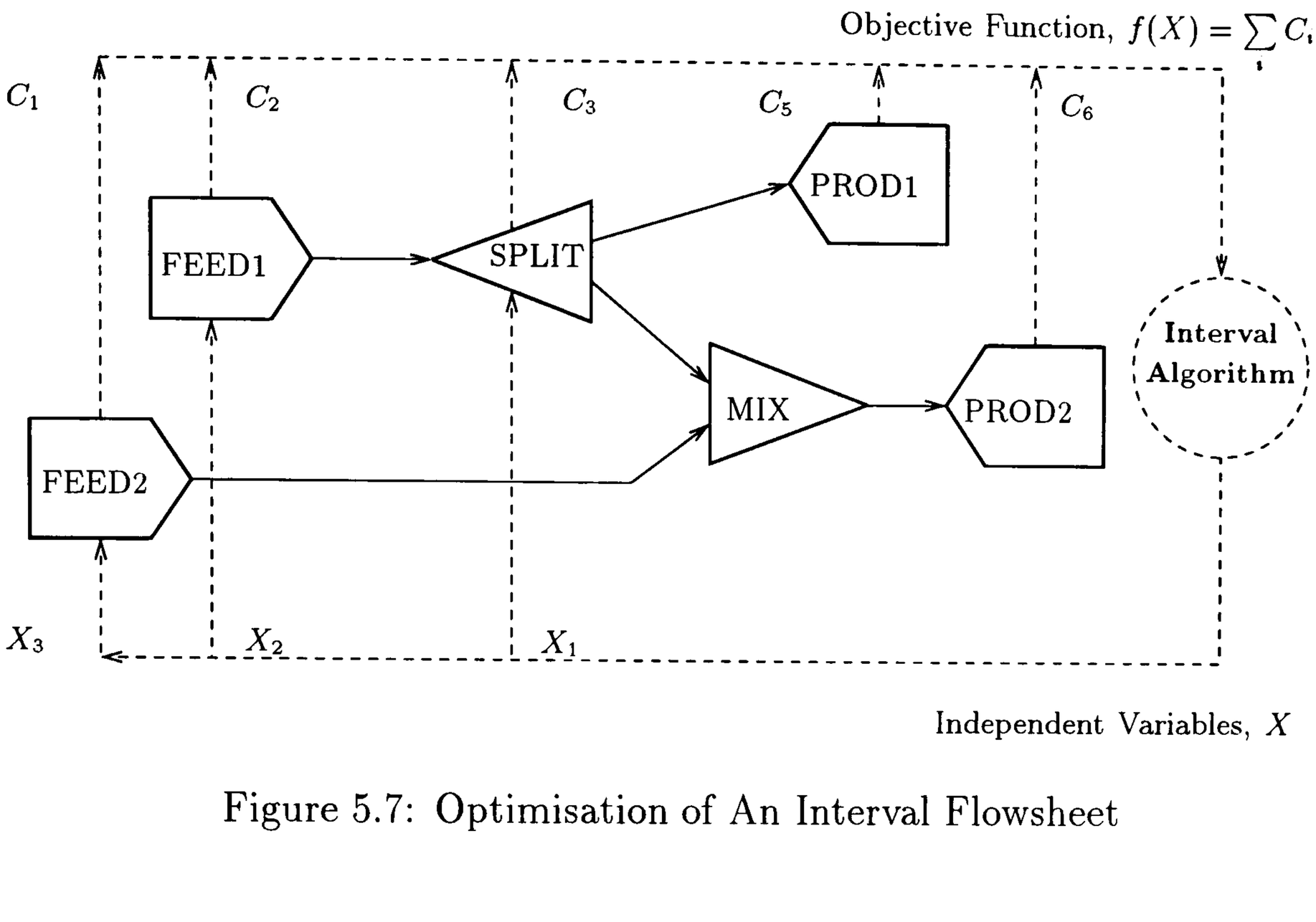


Figure 5.7: Optimisation of An Interval Flowsheet

5.4.2 Optimisation of the Interval Flowsheet

When a flowsheet is built from these general units using interval arithmetic the interval global optimisation algorithm can be applied. The algorithm is provided with initial bounds on the unit parameters which become the initial bound vector X^0. At iteration k, interval parameters, X^k, are chosen by the algorithm. Each unit calculates the output streams and the cost associated with the unit. The summation of these costs provides the objective function. This overall scheme is shown in figure 5.7. The flow of the process streams is shown in full lines and the flow of information through the flowsheet is indicated by dashed lines. The design constraints on the products are added to the optimisation problem and the flowsheet is optimised to minimise the sum of the costs subject to these constraints.

5.4.2.1 Results with Interval Flowsheets

The interval flowsheet described here has been implemented in C++ and solved with the depth-first interval algorithm from §4.1.3. The flowsheet from figure 5.7 can be

Table 5.1: Solution time (s) for different inclusion flowsheets

Algorithm	Figure 5.7	Haverly (Figure 5.1)
Natural Extension	600	-
Mean Value Inclusion	14.4	60

solved using interval analysis in about 600 seconds.

Application of the same techniques to the Haverly Pooling problem (figure 5.1) does not converge to a unique solution in reasonable time. Examination of the list of possible solutions at this point indicates that the design constraints, which are nonconvex with respect to the independent variables, are not fully satisfied.

This is a problem in general for the application of interval optimisation when the solution lies on a nonlinear constraint which cannot be bounded tightly [46]. The problem is made worse because the evaluation through the flowsheet introduces a certain amount of redundant calculations which can make the bounds very conservative.

As noted in [46] the bounds can be improved by using a Mean Value extension of the constraints with optimal centres as described in §3.3.2.1.2. This requires bounds on the gradient of the constraints. The gradient cannot be obtained by inspection because of the modular nature of the flowsheet. Perturbation does not provide bounds on the gradients but construction of a flowsheet with Gradient intervals will provide the necessary information to allow for a Mean Value inclusion. This uses automatic differentiation (see Appendix B.1) with interval arithmetic to provide bounds on the gradients of the flowsheet variables with respect to the independent variables.

Table 5.1 gives the time taken to solve the problem in Figure 5.7 and the Haverly Pooling problem using a natural extension inclusion and the Mean Value inclusion from a gradient bounding flowsheet. The Mean Value form uses the optimal centers from §3.3.2.1.2 to obtain tighter lower bounds at the expense of loosening the upper bounds.

AD as described in §B.1 can be used to obtain the gradients if the unit equations are evaluated in terms of the Gradient type. This means that solution of the interval flowsheet requires an AD flowsheet and a real flowsheet (for the local optimisation step).

5.5 Optimisation of Extended Flowsheets

With the concept of extended types it is possible to see that Interval flowsheets are members of a class of possible models constructed with extended type Modules which use intervals to provide bounds on the outputs, modules based on automatic differentiation which provide gradients for the outputs and, using AD types based on intervals provide bounds on the gradient of the outputs.

Clearly the optimisation algorithm is determined by the type, T, of the units in the flowsheet. As discussed in §5.4 an interval flowsheet can be optimised using an interval algorithm, a flowsheet based on convex underestimators can be optimised by one of the many algorithms based around convex underestimation [25, 52, 55].

Algorithms using convex underestimation are very effective for the solution of equation based models and should result in fewer bisections than an interval method. However, solution of the underestimating NLP requires an evaluation of the flowsheet at each iteration. That is, the convex underestimators must be evaluated at every iteration of the NLP. This is not a difficulty in equation based models where the underestimating model is assumed to have been obtained by hand/symbolic manipulation. The model is fixed with the bounds, X^k, as parameters. In the modular case this advantage is lost. Each evaluation of the underestimating flowsheet *rebuilds* the convex underestimator model and then evaluates it.

This difficulty with convex underestimators arises because the modular flowsheet provides a method of evaluating the underestimators but does not actually provide the underestimators and because general convex underestimators cannot be easily characterised. A linear underestimator can easily be characterised by the coefficients

and a linear model can be solved rapidly using Linear Programming.

5.5.0.2 Linear Underestimation

So far two linear models have been proposed. The NE underestimators from §4.2.2.3 and the MV underestimators from §4.2.2.4. The NE underestimators are linear under/over relaxations which can be used to obtain a linear relaxation of any variable in the flowsheet. Because they are linear they can easily be characterised by the coefficients of the different components and a linear programming problem can be constructed.

The MV relaxation also provides linear underestimators but the linear functions do not 'exist' inside the flowsheet. The underestimators are constructed from gradient information provided by the flowsheet in a manner similar to linearisation of the model. This method has the advantage that one set of gradient bounds obtained by AD can be used to construct as many underestimators as necessary, thereby reducing the total number of AD flowsheet evaluations[1].

Applying the approach taken with the equation based problems in §4.2.2.4 of constructing two underestimating planes at $\underline{x}$ and $\overline{x}$ requires one evaluation of the interval AD flowsheet which provides bounds on the variables in the flowsheet and bounds on the gradients with respect to the independent variables followed by one evaluation with real arithmetic for each of the points $\underline{x}$ and $\overline{x}$. This gives two underestimating planes and two overestimating planes.

In practice the overestimators are not as tight as the underestimators and this tends to hinder convergence because the relaxation of the product constraints are not tight. This suggests that it is better to make some extra evaluations with real arithmetic to obtain a tighter set of overestimators. Using the opposite corners of the box X^k provides a much better upper bound (overestimate) at the expense of extra real evaluations. This is especially advantageous when bilinear terms, $x_1 x_2$, are involved

[1] Each new underestimator requires one evaluation of the real arithmetic flowsheet

Table 5.2: Modular based solution using different relaxations

	MV		MVNE	
	Iter	Time (s)	Iter	Time (s)
IGOR	49	13.30	47	16.70
RIGOR	14	19.97	13	22.42

because the underestimators constructed at $\underline{x_1}, \underline{x_2}$ and $\overline{x_1}, \overline{x_2}$ form the convex hull of x_1x_2 and the overestimators constructed at $\underline{x_1}, \overline{x_2}$ and $\overline{x_1}, \underline{x_2}$ form the concave hull of x_1x_2. Given that the Pooling problem is bilinear in the constraints this means that the extra evaluations required to construct MV underestimators at *each* corner will improve the performance overall because fewer interval AD evaluations need to made. This has been found to be true and the extra planes improve the performance.

5.5.0.3 Results With Extended Type Flowsheets

The results of application to the modular pooling problem using IGOR and RIGOR with MV and MVNE relaxation are given in table 5.2. These results represent an improvement over the interval flowsheets. The times are obtained using the Matlab implementation of (R)IGOR which will, necessarily, be slower than an implementation in C++ . It is difficult to know how much faster a C++ version would be but the fact that the Matlab times for the extended type flowsheets are better than the C++ results using interval flowsheets indicates that the extended flowsheets are substantially better than the interval flowsheet.

The Extended flowsheets are more expensive to evaluate but the improvement in the tightness of the bounds and the reduction in the number of branch variables more than compensates.

Compared to the $1.2s$ taken to solve the Haverly Pooling problem in equation form (5.1) the times are high. This is due to a number of factors;

- Evaluation of quantities through a modular flowsheet requires extra calculation as all intermediate results are calculated.

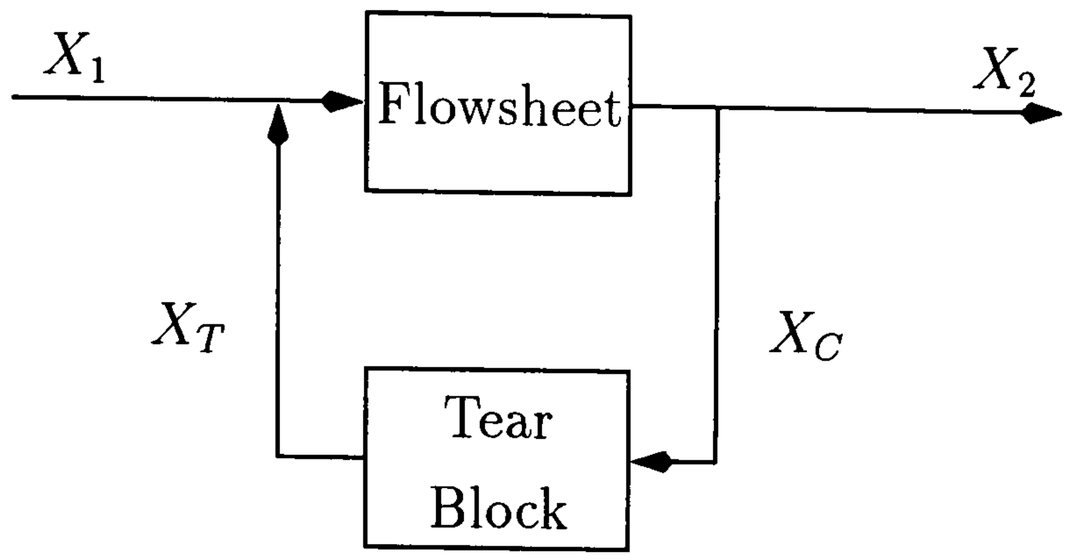


Figure 5.8: Torn Recycle Stream in an Interval Flowsheet

- Evaluation of the flowsheet results in calculation of some quantities, such as stream flowrates, which are not necessary in the EO form.
- In the EO form differentiation is performed symbolically and bounds are generated by interval arithmetic whereas the modular flowsheet uses bounding AD to calculate derivatives at each evaluation.

5.6 Solving the Recycle Problem

It is not generally possible to solve the recycle at each iteration as the solution may not be unique for a given range of input parameters.

If the recycle is torn with a tear block, as shown in figure 5.8, then the range of inputs to the flowsheet, X_1, is selected by the interval optimisation, the flowsheet is calculated through the interval units to produce a set of outputs, X_2, which bound all the possible outputs from the flowsheet with parameters determined by X_1.

Given a 'guess' for the values in the torn stream, X_T, the resulting calculated torn stream, X_C will also be a range which encloses all the possible values of the recycle stream with X_T as an input.

There are two methods of solving this recycle; sequentially and simultaneously.

5.6.1 Sequentially

There may be more than one solution to the recycle problem for the input interval X_1 so it will not, in general, be possible to converge the recycle in the same sense as it is with a flowsheet using real arithmetic.

5.6.1.0.1 Iteration The interval analogue of converging the recycle is to bound all the possible solutions. Given that X_C contains all possible values for the input X_T then $X_T \cap X_C$ is a better guess for X_T. If the intersection is empty then the flowsheet is infeasible for X_1 and X_T.

Thus, it is possible to iterate around the recycle choosing a new, hopefully smaller, range for $X_T = X_T \cap X_C$ with which a new X_C is calculated. Given that it may not possible to obtain equality between X_T and X_C this procedure should probably be stopped when there is no more improvement in the size of X_T.

Because the interval flowsheet must provide bounds which enclose the outputs of the real flowsheet no feasible points should be discarded. This means that the initial guess for X_T must enclose all the possible solutions. In general the solutions are unknown so the choice of this initial interval cannot literally be a guess.

5.6.1.0.2 An Initial Interval for X_T Setting the initial $X_T = [0, +\text{inf}]$ would not exclude any feasible points, if the stream is based on mass flowrates, but will not be particularly useful because it can result in an X_C of infinite width which defeats the object of the iteration sequence.

Given that the algorithm proceeds by successively bisecting an initial interval at $k = 0$ it is not a problem to keep the starting points from previous (parent) input boxes. The task is thus reduced to that of finding X_T for $k = 0$.

One way of obtaining a suitable initial recycle stream is to reverse the usual iteration process and, starting with a single point for X_T enlarge it until $X_T \supseteq X_C$. A (probably better) alternative is to choose an initial X_T based on knowledge of the possible recycle streams from the flowsheet.

The difficulty of choosing an initial range is not restricted to modular flowsheets. It is a problem in all global optimisation algorithms. The sequential nature of the flowsheet means though that the interval can only be reduced using *only* the recycle convergence criterion.

In conclusion an interval flowsheet can, in principle, be operated in a sequential modular mode bounding the solutions to the recycle problem at each iteration of the optimisation algorithm. However, all the problems of slow convergence and nested levels of recycle which are apparent in real arithmetic SM flowsheets are exacerbated by the use of interval analysis because every equation that needs to be solved is an interval equation.

The simultaneous approach gets around some of these problems by allowing the optimisation algorithm to solve more of the equations.

5.6.2 Simultaneously

The simultaneous mode corresponds to using a recycle module where the parameters are independent variables from the optimisation and the residuals are the residuals of the constraint $X_T - X_C = 0$ written as two inequalities.

In this case the recycle is converged as the optimisation converges reducing the amount of computation that must be performed per iteration. However, the physical meaning of the constraint is lost and it is not possible for the optimisation step to make the assumptions that the sequential iteration scheme can make allowing X_T to be set to $X_T \cap X_C$. Though special provision for this case could be made it ties the optimisation method to the flowsheet and blurs the separation between the two which may not be possible or desirable.

When X_T is an optimisation variable it will also be part of the branching scheme which is applied to all optimisation variables which will reduce the size of X_T using the objective function, the recycle constraint and the normal bisection procedure.

5.7 Summary

This chapter has presented methods by which flowsheets can be constructed from modules. The unique thing about these flowsheets is that, by exploiting extended arithmetic types, it is possible to rigorously globally optimise the flowsheet.

These flowsheets use genericity (or polymorphic types) to allow construction of unit models (modules) which can use any Extended type for the underlying computation. This is not to say that any type can be used to optimise the flowsheet but that generic models can be used to obtain whatever information is appropriate for a given type. This has the advantage that a single code for each model can be used to generate floating point results, interval results, gradient information and gradient bounding information.

It might be more accurate to refer to this construction as an object-oriented flowsheet, it employs ideas from OO such as encapsulation (details of unit calculations are hidden) and polymorphism (each arithmetic type determines the manner in which results are calculated). In some sense the general unit module proposed in this chapter is a base (or parent) of the modules which will specialise this general behaviour.

All of these factors make it easier to construct modular flowsheets which can be globally optimised but they are not necessary for it to be possible. A different flowsheet for each of the types could be used and the evaluations made through this flowsheet. The important aspect is that interval analysis and automatic differentiation can be combined to propagate gradient bounding information through a flowsheet of connected modules and provide bounds on the outputs (or intermediate quantities) of a modular flowsheet.

Chapter 6

Conclusion

This thesis has addressed the problem of using interval based global optimisation algorithms to solve the widest possible range of continuous optimisation problems. The approach is developed in such a way that it can be applied, effectively, to constrained problems exploiting information about the structure of the problem.

Chapter 5 developed a technique for constructing modular systems which can be globally optimised.

6.1 Global Optimisation Algorithms

The development of the global optimisation approach presented here stems from the interpretation of interval bounding rules as a bound constrained linear program. This puts the interval method on a more even footing with other methods which solve relaxed problems. The relaxed problem formulation allows convex constraints from NCP to be included in the lower bounding procedure.

Further reformulation of the lower bounding problem is used to increase the number of constraints and terms in the objective function which can be retained in the lower bounding problem. This reformulation opens the possibility of adding more

complementary relaxations to the lower bounding problem such as the MV and NE relaxations which are derived from the Mean Value and Natural Extension forms of interval arithmetic. The use of interval inclusion and MV relaxation retains the wide applicability of the interval methods with tighter formulation for differentiable terms.

The MV relaxation is more general and more widely applicable than the α-BB relaxation providing a linear relaxation of any differentiable term in the problem by using interval analysis to bound the gradient. The linear relaxation is tight at the corners of the interval and will approximate the problem better as the interval is reduced, as long as the gradient bounds do not increase.

Though it is possible to construct convex relaxations using these techniques we have chosen to use only linear relaxations of NCP which can be solved efficiently using linear programming. The decision to use linear relaxations is based on the need to solve relaxed problems efficiently and reliably and in preparation for the modular systems described in chapter 5.

The relaxation techniques are applied in two algorithms, IGOR & RIGOR, which employ a depth-first branch and bound search to rigorously locate an ϵ-global minimiser. The RIGOR algorithm augments the IGOR approach with a reduction step which refines the bounds on a given partition using the upper bound and relaxed constraint space. For the set of test problems attempted the use of RIGOR with the MVNE relaxation is the best of the approaches tested.

6.1.1 Future Work

The performance of the standard interval algorithms which apply interval analysis to the problem directly is poor by comparison to the methods which apply interval analysis indirectly. In α-BB, interval techniques are used to solve an interval polynomial to construct underestimators, in the work of Epperly [27] an interval linear program is solved to construct quadratic underestimators and in IGOR the interval analysis is applied to bound the gradient and construct underestimators.

These techniques all exploit the structure of the problem before applying interval analysis. Tighter bounds are obtained by applying interval arithmetic to some problem other than bounding the objective or constraints. They also all rely on the application of local algorithms which can be very efficient and are also quite widely available.

For the constrained NLPs arising in Process Engineering applications it is vital that the structure of the problem be exploited and if interval analysis is to be pursued it will be as a part of an algorithm and not as the basis for it.

The cost of using local optimisation to get lower bounds, that is, the cost of applying interval analysis indirectly is that some of the useful verification properties of interval analysis are lost and rounding errors due to floating point arithmetic are not accounted for.

Some might judge this a very serious limitation and some may not. In the case of IGOR and RIGOR it should be possible to regain the property of verified solution. This would hopefully result in an algorithm which retains the generality of interval methods, the efficiency of relaxation methods and provides verified solutions.

The results comparing RIGOR and IGOR indicate that RIGOR is much better for problems with loose initial bounds but that it performs more tightening steps than are strictly necessary. A better algorithm would look at the change in the constraint matrix for the lower bounding problem and tighten variables that appear in constraints which have tightened since the box was partitioned. Such an algorithm has been developed but the results were not ready for publication here so they will be presented at a later date.

The use of linear relaxations has been investigated. Clearly a convex relaxation (one retaining convex terms) would be superior for problems where only the convex constraints are active at the solution. The question which needs to be addressed is whether the tighter bounds justify the additional expense of using an NLP algorithm to solve relaxed problems. Also, it is difficult to see how the reduction steps can be efficient with nonlinear constraints and the simpler reduction techniques proposed

for nonlinear relaxations would need to be used.

If a nonlinear relaxation was to be used then some quadratic underestimators may also improve performance. These might be derived from an interval Hessian or from the quadratic interval forms given by Tupper [62].

Some mixed integer problems can already be solved using the approach presented here but the algorithm needs to be extended if general MINLP problems are to be solved. There are two main options for such an implementation: to use IGOR to solve NLPs within a mixed integer branch and bound approach or to incorporate the integer branch and bound into the global branch and bound procedure.

6.2 Optimisation of Modular Systems

Although global optimisation has been applied to equation based flowsheet problems there have not been any previous attempts to apply global optimisation to modular flowsheets.

It is not possible guarantee global optimality of a true black-box flowsheet based on floating point arithmetic but this thesis has developed a method for constructing systems from modules whose expressions are known. This method is based on the use of Extended Arithmetic types and generic programming to communicate necessary information through the flowsheet.

A general unit module is proposed which allows various different constructions. The general unit can be used to receive input streams from preceding units and operating parameters from the optimisation algorithm. It then calculates outputs which are passed to the next unit. Each unit also has an operating cost (possibly zero) and a vector of constraint residuals which can be used to satisfy certain quality requirements or implicit equations which need to be solved in the unit.

This general approach allows the flowsheet to be operated in sequential or simultaneous modes or some appropriate hybrid of the two. In the case of the interval

flowsheet a scheme for operating it in sequential mode has been proposed where only product units have residuals to be satisfied in the optimisation algorithm, all other solutions are made within the flowsheet. It is not clear how general extended types can be used in this form. In particular we are not aware of any results relating to the convergence of interval gradients in iterative procedures.

Because sequential operation requires solution of interval equations and multiple iteration loops, simultaneous solution of implicitly defined properties has been recommended. This provides some of the flexibility of an equation based approach but each equation is still associated with a unit so it is easier to diagnose problems.

The use of polymorphic/generic arithmetic types means that much more information about the unit model can be obtained without exposing the equations of the model in the software. This means that new models can be added without recompilation and without exposing details of unit models which may represent some competitive advantage.

The flowsheet is constructed and floating point, interval and interval gradient types (using automatic differentiation) are used to construct a linear relaxation which can be solved using IGOR or RIGOR. The use of a linear relaxation means that the solution of the relaxed problem occurs outside the flowsheet because the LP can be characterised and evaluated without re-evaluating the flowsheet. This would not be possible with general convex relaxations which would require a flowsheet evaluation at every iteration of the relaxed NLP.

The time required to solve the modular problem is higher than the equation based formulation but not prohibitively so for the examples attempted. The time saved formulating and debugging the problem more than compensates.

6.2.1 Future Work

We have demonstrated that acyclic modular systems can be constructed so that global optimisation algorithms can be applied. Two schemes for solving recycle

problems have been outlined but no computational experience has been presented for these techniques.

Operating the interval flowsheet in sequential mode is, in principle, feasible but the efficiency of the interval flowsheet is not promising. Solution of the more general Extended Flowsheets in sequential mode has not been considered as each Type will require a different solution method. Work is currently underway to solve recycle problems in a simultaneous mode using RIGOR with Gradient bounding types.

We have not addressed the problem of incorporating thermodynamic information into a flowsheet (modular *or* equation based). The development of thermodynamic property code which uses Extended Types would provide both flowsheeting paradigms with property prediction.

The use of Extended Types and genericity in construction of the flowsheet means that new Types can be used without redesigning the flowsheet. This means that other algorithms could also be applied. A Hessian Type could be used to derive α-BB underestimators, convex underestimators could be used to apply McCormick's approach and the generalised interval types, such as quadratic, could also be used.

As we have noted, the use of generic unit operations has the same effect, in principle, of using polymorphic Extended Types. Using polymorphic types provides greater flexibility as types can then be dynamically linked and no recompilation is necessary for adding a new type or a new unit model. However, this flexibility comes at a cost in terms of speed. If only a few Extended Types were to be used the generic version works well. If a flowsheeting package is designed for many extended types to be used then the polymorphic approach should be investigated.

Process flowsheets are only one type of modular system. There are many others including software, electronic systems, computer design and many kinds of network problems. The techniques employed here to design Extended Type process flowsheets could also be applied in these other areas to allow global optimisation or to pass other extended information around modular systems.

Bibliography

[1] J.D Perkins, I.D.L Bogle (1988) "Numerical Methods for the Simulation of Chemical Plant" Computational Techniques and Applications, J Noye & C Fletcher (eds). Elsevier Science, North Holland.

[2] C Schmid, L.T Biegler (1994) "A Simultaneous Approach for Flowsheet Optimization with Existing Modeling Procedures" Trans. IChemE. **72** 382–388.

[3] A Törn, A Žilinskas (1989), "Global Optimization. Lecture Notes in Computer Science." Springer–Verlag, Berlin.

[4] A.H.G.R Kan, G.T Timmer (1987), "Stochastic Global Optimization Methods 2. Multi-Level Methods." Math. Prog. **39** (1) 57–78.

[5] J.H Holland (1975) "Adaptation in Natural and Artificial Systems." University of Michigan Press, Ann Arbor.

[6] L Chambers (Ed) (1995) "Practical Handbook of Genetic Algorithms, Volume 1 Applications" CRC Press Inc. Boca Ranton, Florida.

[7] I.P Androulakis, V Venkatsubramanian (1991) "A Genetic Algorithmic Framework for Process Design and Optimization". Computers chem. Engng. **15** 217–228 (4)

[8] A.O Griewank (1981) "Generalised Descent For Global Optimisation." J. Optim. Theory. Appl. **34** (8) 11–39.

[9] R.P Ge (1983) "A Filled Function Method for Finding a Global Minimiser." Dundee Biennial Conference on Numerical Analysis (3).

[10] R.P Ge (1990) "A Filled Function Method for Finding a Global Minimiser of a Function of Several Variables." Math. Prog. **46** (2) 191–204.

[11] R.P Ge, Y.F Qin (1987) "A Class of Filled Functions for Finding Global Minimisers of a Function of Several Variables." J. Optim. Theory. Appl. **54** (2) 241–252.

[12] A.V Vilkov, N.P Zhidkov, Shchedrin (1975) "A Method of Search for the Global Optimum of a Function of One Variable" Journal of Comput. Math. and Math. Phys. **15** 1040–1042 (3).

[13] A.V Levy, A Montalvo (1985) "The Tunnelling Algorithm for Global Minimisation of Functions." SIAM J, Sci. Stat. Comp. **6** 15–29 (3,8).

[14] T.F Edgar, D.M Himmelblau (1988) "Optimization of Chemical Processes", Chemical Engineering Series, McGraw-Hill International, New York.

[15] R Horst (1986) "A General Class of Branch and Bound Methods in Global Optimization with Some New Approaches to Concave Minimization." J. Opt. Theory Applic. **51** (2) 271–291

[16] A.M Geoffrion (1972), "Generalized Benders Decomposition." J. Opt. Theory Applic. **10** (273)

[17] C.A Floudas, V Visweswaran (1990) "A Global Optimization Algorithm (GOP) for Certain Classes of Nonconvex NLPs. 1 Theory." Computers chem. Engng. **14** (12), 1397–1417.

[18] C.A Floudas, A Aggarwal, A.R Ciric (1989) "Global Optimum Search for Nonconvex NLP and MINLP Problems." Computers. chem. Engng. **13** (10) 1117–1132.

[19] V Manousiouthakis (1991) "On the Generalized Benders Decomposition" Computers chem. Engng. **15** (10), 691–700.

[20] N.V Sahinidis, I.E Grossmann (1991) "Convergence Properties of the Generalized Benders Decomposition" Computers chem. Engng. **15** (12), 481–491.

[21] W.B Liu, C.A Floudas (1993) "A Remark On The GOP Algorithm For Global Optimization." J. Global Optimization **3** (4) 519-521.

[22] V Visweswaran, C.A Floudas (1996) "New Formulations and Branching Strategies for the GOP Algorithm." 75–111 in I.E Grossmann (ed.) "Global Optimisation for Engineering Design." Kluwer Academic, Publishers, Dordrecht.

[23] J.E Falk, R.M Soland (1969). "An Algorithm for Separable Nonconvex Programming Problems." Management Sci. **15** (9) 550–569.

[24] R.M Soland (1971). "An Algorithm for Separable Nonconvex Programming Problems II: Nonconvex Constraints." Management Sci. **17** (11) 759–773.

[25] G.P McCormick (1976) "Computability of Global Solutions to Factorable Nonconvex Programs: Part 1 - Convex Underestimating Problems." Math. Prog. **10** 147–175

[26] T.G.W Epperly (1995) "Global Optimization of Nonlinear Nonconvex Programs using Parallel Branch and Bound" Ph.D Thesis, University of Wisconsin – Madison.

[27] T.G.W Epperly, R.E Swaney (1996) "Branch and Bound for Global NLP." 1–35 in I.E Grossmann (ed.) "Global Optimisation for Engineering Design." Kluwer Academic, Publishers, Dordrecht.

[28] C.S Adjiman, I.P Androulakis, C.D Maranas and C.A Floudas (1996) "A global Optimisation Method, αBB, for Process Design" Computers chem. Engng. **20S** S419–S424.

[29] S.A Piyavskii (1972). "An Algorithm for Finding the Absolute Extremum of a Function." USSR Comp. Mat. & Mat. Phys. **12** 57–67.

[30] B.O Shubert (1972) "A Sequential Method of Seeking the Global Maximum of a Function" SIAM J. on Numerical Analysis **9** 379–388.

[31] C.C Meewella, D.Q Mayne (1988) "An Algorithm for Global Optimization of Lipschitz Continuous Functions." J.Optim. Theory. Appl **57** (2) 307–322.

[32] R.H Mladineo (1992) "Convergence Rates of a Global Optimisation Algorithm" Math. Prog. **54** 223–232.

[33] P Hansen, B Jaumard, S.H Lu (1992). "On Using Estimates of Lipschitz-Constants in Global Optimization." J. Optim. Theory. Appl. **75** (1) 195–200.

[34] P Hansen, B Jaumard, S.H Lu (1992). "Global Optimization of Univariate Lipschitz Functions. 1. Survey and Properties ." Math. Prog. **55** (3) 251–272.

[35] R.E Moore (1966), "Interval Analysis.", Prentice-Hall, Englewood Cliffs, NJ.

[36] H Ratschek, J Rockne (1988), "New Computer Methods for Global Optimization", Ellis Horwood Ltd., Chichester, West Sussex, England.

[37] M. Rosenlicht (1968) "Introduction to Analysis" Dover Publications Inc. New York.

[38] E Baumann (1986), "Optimal Centered Forms", *Freiburger Intervall-Berichte* 86/6, Institut für Angewandte Mathematik, Universität Freiburg.

[39] L.B Rall (1981) "Automatic Differentiation: Techniques and Applications." Lecture Notes in Computer Science, Springer-Verlag Berlin.

[40] E Hansen (1992) "Global Optimization Using Interval Analysis." Marcel Dekker, New York.

[41] E.R Hansen (1980), "Global Optimisation using Interval Analysis – the multi-dimensional case" Numer. Math. **34** 247–270.

[42] R.B Kearfott (1996). "Test Results for an Interval Branch & Bound Algorithm for Equality-Constrained Optimization." 181–199 In C.A Floudas, P.M Pardalos (eds.) State of the Art in Global Optimization, Kluwer Academic Publishers, the Netherlands.

[43] R Vaidyanathan, M El-Halwagi (1994) "Global Optimisation of Nonconvex Programs via Interval Analysis" Computers chem. Engng. **18** 889.

[44] C.A Schnepper, M.A Stadtherr (1993) "Application of a Parallel Interval Newton/Generalized Bisection Algorithm to Equation-Based Chemical Process Flowsheeting" Interval Computations **4** 40–64.

[45] A.P Leclerc (1992) "Efficient and Reliable Global Optimization" PhD thesis Ohio State University.

[46] R.P Byrne and I.D.L Bogle (1996) "Global Optimisation using Interval Analysis." 155–174 in I.E Grossmann (ed.) "Global Optimisation for Engineering Design." Kluwer Academic, Publishers, Dordrecht.

[47] E.M.B Smith, C.C Pantelides (1996) "Global Optimisation of General Process Models." 355–386 in I.E Grossmann (ed.) "Global Optimisation for Engineering Design." Kluwer Academic, Publishers, Dordrecht.

[48] R.P Byrne, I.D.L Bogle (1995) "Partitioning in Interval Methods for Global Optimisation." Proc. IChemE Research Event 1995 **1** 98–100.

[49] R.B Kearfott (1996). Personal communication.

[50] R.P Byrne, I.D.L Bogle (1995) "An Accelerated Interval Method for Global Optimisation." Computers chem. Engng. **20** S49–S54.

[51] R Vaidyanathan, M El-Halwagi (1996) "Global Optimisation of MINLPs by Interval Analysis." 175–195 in I.E Grossmann (ed.) "Global Optimisation for Engineering Design." Kluwer Academic, Publishers, Dordrecht.

[52] H.S Ryoo and N.V Sahinidis (1995) Global Optimization of Nonconvex NLP s and MINLPs with Applications in Process Design. Computers chem. Engng. **19** (5) 551–566.

[53] P Hansen, B Jaumard, S.H Lu (1991). "An Analytical Approach to Global Optimization" Math. Prog. **52** 227–254.

[54] E.M.B Smith (1996) "On the Optimal Design of Continuous Processes" PhD thesis, University of London.

[55] I Quesada, I.E Grossmann (1995). "Global Optimization of Bilinear Process Networks with Multicomponent Flows" Computers chem. Engng **19** (12) 1219–1242.

[56] M. Manousiouthakis, D. Sourlas (1992) "A Global optimization approach to Rationally Constrained Rational Programming" Chem. Engng. Communications. **115** 127–147.

[57] C.A Floudas, P.M Pardalos (1990) "A Collection of Test Problems for Constrained Global Optimization Algorithms." Lecture Notes in Computer Science, Springer-Verlag, Berlin.

[58] G Stephanopoulos, A.W Westerberg (1975) "The use of Hestenes' Method of Multipliers to Resolve Dual Gaps in Engineering Systems Optimization" J. Optim. Theory Applic. **15** 285–309.

[59] I Quesada, I.E Grossmann (1993) "Global Optimization Algorithm for Heat-Exchanger Networks." Ind. Eng. Chem. Res. **32** (3) 487–499.

[60] L.T Biegler (1983) "Simultaneous Modular Simulation and Optimization" Foundations of Computer-Aided Process Design, Proceedings of the Second International Conference held at Snowmass, Colorado.

[61] J.D Perkins (1983) "Equation Oriented Flowsheeting" Foundations of Computer Aided Process Design, Proceedings of the Second International Conference held at Snowmass, Colorado.

[62] J.A Tupper (1996) "Graphing Equations with Generalized Interval Arithmetic" MSc thesis, University of Toronto.

[63] J.L.D Comba, J Stolfi (1993) "Affine Arithmetic and its Applications to Computer Graphics" Anais do VII Sibgrapi, preprint from `http://www.doc.unicamp.br/~stolfi`.

[64] R.P Byrne, I.D.L Bogle (1997) "Global Optimisation for Modular Process Flowsheet Optimisation " Proc. IChemE Research Event 1997 **2** 701–704.

[65] P.M Pardalos, J.B Rosen (1987) "Constrained Global Optimization: Algorithms and Applications" Lecture Notes in Computer Science, Springer-Verlag, Berlin.

[66] C.A. Schnepper, M. A. Stadtherr (1996) "Robust Process Simulation Using Interval Methods" Computers chem. Engng. **20** 187–199.

[67] G.R Kocis, I.E Grossmann (1991). "Global Optimization of Nonconvex Mixed–Integer Nonlinear–Programming (MINLP) Problems in Process Synthesis."Ind. Eng. Chem. Res. **27** (8) 1407–1421.

[68] R Horst, H Tuy (1992) "Global Optimization. Deterministic Approaches" Springer-Verlag, Berlin.

Appendix A

Mathematical Definitions

A.1 Relevant Properties of Sets and Functions

A.1.1 Continuity

A function $f(x)$ is said to be continuous at $a \in \mathbb{R}$ if $f(x) \to a$ as $x \to a$ this, of course, requires that the limit exists. Another definition which illustrates all the aspects in which continuity can fail, is

$$\lim_{x \to a^-} f(x) = \lim_{x \to a^+} f(x) = f(a). \tag{A.1}$$

A.1.2 Convexity

The property of convexity, both in sets and functions, is very important to optimisation. A function $f(x)$ is said to be convex on a convex set $\mathcal{S}$ if for every $x_1, x_2 \in \mathcal{S}$ and γ real, $0 \leq \gamma \leq 1$,

$$f\left(\gamma x_1 + (1-\gamma)x_2\right) \leq \gamma f(x_1) + (1-\gamma) f(x_2) \tag{A.2}$$

Geometrically this means that a line joining any two points on the graph of $f(x)$ being above the graph. A function $f(x)$ is concave if, and only if, $-f(x)$is convex. $f(x)$ may be said to be quasiconvex if

$$f\left(\gamma x_1 + (1-\gamma)x_2\right) \leq \max\left(f(x_1), f(x_2)\right). \tag{A.3}$$

A set $\mathcal{S} \subset \mathbb{R}^n$ is said to be convex if for every $x_1, x_2 \in \mathcal{S}$ and γ real, $0 \leq \gamma \leq 1$,

$$\gamma x_1 + (1-\gamma)x_2 \in \mathcal{S}. \tag{A.4}$$

Geometrically this states that a line joining two points in $\mathcal{S}$ is completely within $\mathcal{S}$. An important property of convex sets is illustrated by the following problem:

$$\min_{x \in \mathcal{P}} f(x) \tag{A.5}$$

where $\mathcal{P} \subseteq \mathbb{R}^n$ is a compact, convex set and $f(x)$ is continuous on $\mathcal{P}$.

This problem is called a convex/nonconvex/concave programming problem according to the behaviour of $f(x)$ on $\mathcal{P}$. When the objective function is convex the local minimum is also global, similarly for maximising a concave function over a convex set. This, of course, is not necessarily true when either $f(x)$ or $\mathcal{P}$ are nonconvex

A.1.3 Convex Hull/Envelope

The convex hull of a set $\mathcal{S}$ is the smallest convex set which contains the extreme points of $\mathcal{S}$. This means that $\mathcal{S}$ and its convex hull, $e_{\mathcal{S}}$ share the same extreme points.

The convex envelope, $e_f(x)$ of a function $f : \mathcal{S} \to \mathbb{R}$, where $\mathcal{S}$ is a nonempty subset of $\mathbb{R}^n$, is a function such that

1. $e_f(x)$ is convex on the convex hull of $\mathcal{S}$

2. $e_f(x) \leq f(x)$ for all $x \in \mathcal{S}$

3. if $c_f(x)$ is any convex function defined on $e_{\mathcal{S}}$ such that $c_f(x) \leq f(x)$ for all $x \in e_{\mathcal{S}}$ then $c_f(x) \leq e_f(x)$ for all $x \in e_{\mathcal{S}}$.

A.1.4 Lipschitz Condition

A real valued function $f(x)$ obeys a Lipschitz condition on $\mathcal{S}$ if there is a constant L such that,

$$|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\| \quad \forall x_1, x_2 \in \mathcal{S}. \tag{A.6}$$

Clearly, if $f(x)$ is Lipschitz with constant L it is also for all $L' \geq L$.

The case that $f(x)$ in (NCP) is Lipschitz is a very common one. However, computationally the constant L is not always easily determined. It is, also, often reasonable to assume that some of the derivatives of $f(x)$ are Lipschitz. Indeed, a function with continuous bounded derivatives is Lipschitz, and L can be determined.

Appendix B

Extended Arithmetic Types

B.1 Automatic Differentiation

Automatic Differentiation, AD, is a method for simultaneously computing partial derivatives with a function's value by replacing the numbers in a function with a multicomponent object, called Gradient, whose algebraic properties incorporate the chain rule of differentiation.

AD lies somewhere between finite difference differentiation and symbolic or analytical differentiation. It is considerably more reliable and accurate than finite difference methods [39] but is not a symbolic method as it calculates partial derivatives for a single value of the independent variables. It is usually less efficient than hand tuned analytical derivatives but often more efficient than either finite or symbolic methods.

AD is less general than finite difference methods because it cannot be used on 'black box' models but it is more general than symbolic methods because the expression may be produced by a number of procedures. This is not a concern when the application is global optimisation which cannot be used to rigorously solve such models anyway.

The gradient composite object is a pair $g = (v, v')$ with $v \in \mathbb{R}, v' \in \mathbb{R}^n$ where n is the number of independent variables. For $n = 1$, if g is the independent variable then $v' = 1$, if g is a constant $v' = 0$. For the multivariate case the v' part of the ith variable is the ith Cartesian basis vector.

The common arithmetic operations are determined using the chain rule. For example if $u = u(x)$ and a is a constant

$$\begin{aligned} y &= u, & \frac{\partial y}{\partial x} &= \frac{\partial u}{\partial x} \\ y &= au, & \frac{\partial y}{\partial x} &= a\frac{\partial u}{\partial x} \\ y &= u^2, & \frac{\partial y}{\partial x} &= 2u\frac{\partial u}{\partial x} \end{aligned} \tag{B.1}$$

etc...

Thus arithmetic operations on objects of type AD can be defined.

$$\begin{aligned} (u, u') + (v, v') &= (u + v, u' + v') \\ (u, u') - (v, v') &= (u - v, u' - v') \\ (u, u') * (v, v') &= (uv, uv' + vu') \\ (u, u')/(v, v') &= (u/v, (vu' - uv')/v^2) \end{aligned} \tag{B.2}$$

Other functions, such as $\ln x$, can easily be calculated from the chain rule.

A major advantage of AD is that it can be applied to factorable functions. That is, the function, F, may be written as a set of procedures and not as a single string. This relies on the fact that every function, no matter how complicated, is executed on a computer as a sequence of elementary operations and elementary functions. By repeated application of the chain rule to the composition of these elementary operations it is possible to compute derivatives that are accurate to machine precision. The procedures involved may contain arbitrary branches, conditional loops and subprocedures [39].

B.2 Linear Underestimator Arithmetic

In general, if on some interval $X \in \mathbb{I}^n$

$$\begin{array}{rcccl} \underline{u} & \leq & u(x) & \leq & \overline{u} \\ \underline{v} & \leq & v(x) & \leq & \overline{v} \end{array} \tag{B.3}$$

and

$$\begin{array}{rcccl} c_u(x) & \leq & u(x) & \leq & C_u(x) \\ c_v(x) & \leq & v(x) & \leq & C_v(x) \end{array} \tag{B.4}$$

where $c_z(x)/C_z(x)$ are linear under/overestimators of $z(x)$ on X,

$$\begin{array}{rcl} c_v(x) & = & \alpha_v^T x + a \\ C_v(x) & = & \beta_v^T x + b \end{array} \tag{B.5}$$

then the following results allow calculation of linear over/underestimators for addition and subtraction of $u(x)$ and $v(x)$.

$$\begin{array}{rcccl} c_u(x) + c_v(x) & \leq & u(x) + v(x) & \leq & C_u(x) + C_v(x) \\ c_u(x) - C_v(x) & \leq & u(x) - v(x) & \leq & C_u(x) - c_v(x). \end{array} \tag{B.6}$$

Multiplication is somewhat more complicated. Say $0 \notin [\underline{v}, \overline{v}]$, then

$$\begin{array}{rcccll} \underline{u} c_v(x) & \leq & u(x)\, v(x) & \leq & \overline{u} C_v(x) & \text{if } \underline{v} \geq 0 \\ \\ \overline{u} c_v(x) & \leq & u(x)\, v(x) & \leq & \underline{u} C_v(x) & \text{if } \overline{v} \leq 0 \end{array} \tag{B.7}$$

If both $[\underline{u}, \overline{u}]$ and $[\underline{v}, \overline{v}]$ cross zero then, using the convex hull of $f_u(x)\, v(x)$

$$\begin{array}{rcl} u(x)\, v(x) & \geq & \underline{u} C_v(x) + \underline{v} C_u(x) - \underline{u}\, \underline{v} \\ \text{and} & & \\ u(x)\, v(x) & \geq & \overline{u} c_v(x) + \overline{v} c_u(x) - \overline{u}\, \overline{v} \end{array} \tag{B.8}$$

The final development is to obtain similar bounding rules for terms of the form $T(z)$ where T is a function of a single variable (such as $\ln z, z^2$ or $1/z$) and $z = U(x)$. This information can be calculated for a continuous function, T, of a single variable, z, if the linear envelopes

$$\begin{array}{rcccl} c_T(z) & \leq & T(z) & \leq & C_T(z) \\ c_v(x) & \leq & v(x) & \leq & C_v(x) \end{array} \tag{B.9}$$

of $T(z)$ on $Z = [\underline{z}, \overline{z}]$ and $v(x)$ on X are known, where

$$c_T(x) = \alpha_T^T x + a_T \tag{B.10}$$

We have,

$$T(v(x)) \geq c_T(v(x)) \geq \left\{ \begin{array}{ll} c_T(c_v(x)) & \text{if} \quad \alpha_T \geq 0 \\ c_T(C_v(x)) & \text{if} \quad \alpha_T < 0. \end{array} \right\} \tag{B.11}$$

which can be used to underestimate $T(v(x))$. Overestimators can be obtained in a similar manner.

Given that the bounds, $\underline{u}, \overline{u}, \underline{v}, \overline{v}$ can be obtained through interval analysis these rules can be used to create an extended type which provides linear under/over estimators for factorable formulations.

Alternative linear arithmetic forms can be found in [62] and 'Affine arithmetic', a linear form of interval arithmetic where the upper and lower bounding functions are parallel, can be found in [63].

Tupper also presents quadratic and polynomial interval forms [62].

Appendix C

Selected Test Problems

Problem **camel**, 'Six Hump Camel Back Function', from [36],

$$\min_x \quad 4x_1^2 - 2.1x_1^4 + \tfrac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \tag{C.1}$$

with $X^0 = [(-10, 5), (-7, 4)]$. The problem has 15 stationary points in the region of interest. The two global minima are at $[0.0898, -0.7126]^T$ and $[-0.0898, 0.7126]^T$.

Problem **gos2** from [18],

$$\begin{aligned} \min_x \quad & -x_1 + x_1x_2 - x_2 \\ \text{s.t.} \quad & \\ & -6x_1 + 8x_2 \leq 3 \\ & 3x_1 - x_2 \leq 3 \end{aligned} \tag{C.2}$$

where $x_1 \geq 0$ and $x_2 \leq 5$. The problem has two minimisers at $[0.916, 1.062]$ and $[1.167, 0.5]$ with $f(x) = -1.0052$ and -1.0833 respectively.

Problem **gop3** from [17];

$$\begin{aligned} \min_x \quad & -x_1 - x_2 \\ \text{s.t.} \quad & \\ x_1 x_2 &\leq 4 \\ 0 \leq x_1 &\leq 4 \\ 0 \leq x_2 &\leq 8 \end{aligned} \tag{C.3}$$

The global solution is -8.5 with $x^* = (8, 0.5)$.

Problem **ssb** from [54],

$$\begin{aligned} \min_x \quad & x_3 \\ \text{s.t.} \quad & \\ x_1 + x_2 - x_4 &= 0 \\ x_1\, x_2 + x_3 &= 0 \end{aligned} \tag{C.4}$$

where X^0 is given by

$$\begin{aligned} 0 \leq x_1 \leq 50 \qquad & 0 \leq x_2 \leq 50 \\ -100 \leq x_3 \leq 100 \qquad & -100 \leq x_4 \leq 100. \end{aligned}$$

The global minimum is -4 at $x^* = (2, 2, -4, 4)$, there is a local solution at the origin.

The equation based formulation of problem **haverly** from [55] is

$$\begin{aligned} \min_{F, x_A} \quad & 6F_1 + 16F_2 + 10F_4 - 9F_5 + 10F_7 - 15F_8 \\ \text{s.t.} \quad & \\ F_1 + F_2 - F_3 - F_6 &= 0 \\ F_3 + F_4 - F_5 &= 0 \\ F_6 + F_7 - F_8 &= 0 \\ 0.03F_1 + 0.01F_2 - F_3 x_A - F_6 x_A &= 0 \\ F_3 x_A + 0.02F_4 - 0.025F_5 &\leq 0 \\ F_6 x_A + 0.02F_7 - 0.015F_8 &\leq 0 \end{aligned} \tag{C.5}$$

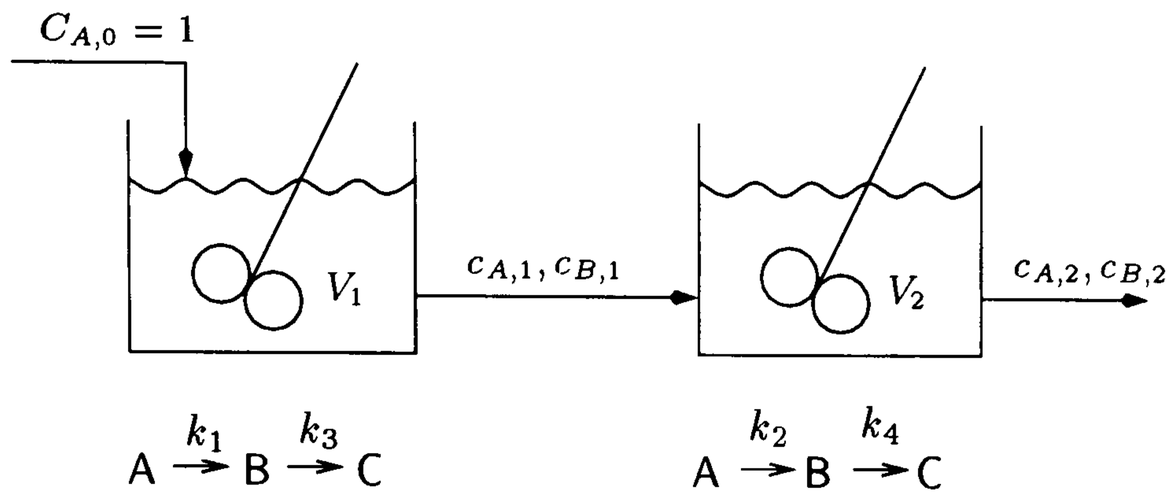


Figure C.1: Reactor Network Optimisation

The global minimiser is

$$F = [0, 100, 0, 0, 0, 100, 100, 200]^T$$

and $x_A = 0.01$ with $f(x) = -400$.

Problem **ryoo20** a reactor network design problem from [52, 56] shown in Figure C.1,

$$\begin{aligned} \min_x \quad & -x_4 \\ \text{s.t.} \quad & \\ x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 &= 0 \\ x_1 - 1 + k_1 x_1 x_5 &= 0 \\ x_2 - x_1 + k_2 x_2 x_6 &= 0 \\ x_3 + x_1 - 1 + k_3 x_3 x_5 &= 0 \\ x_5^{0.5} + x_6^{0.5} &\leq 4 \end{aligned} \tag{C.6}$$

where $0 <= x <= (1, 1, 1, 1, 16, 16)$ and

$$\begin{aligned} k_1 &= 0.09755988 & k_2 &= .99k_1 \\ k_3 &= 0.0391908 & k_4 &= 0.9k_3. \end{aligned}$$

The solution is $x^* = [0.771462, 0.516997, 0.204234, 0.388812, 3.036504, 5.096052]^T$.

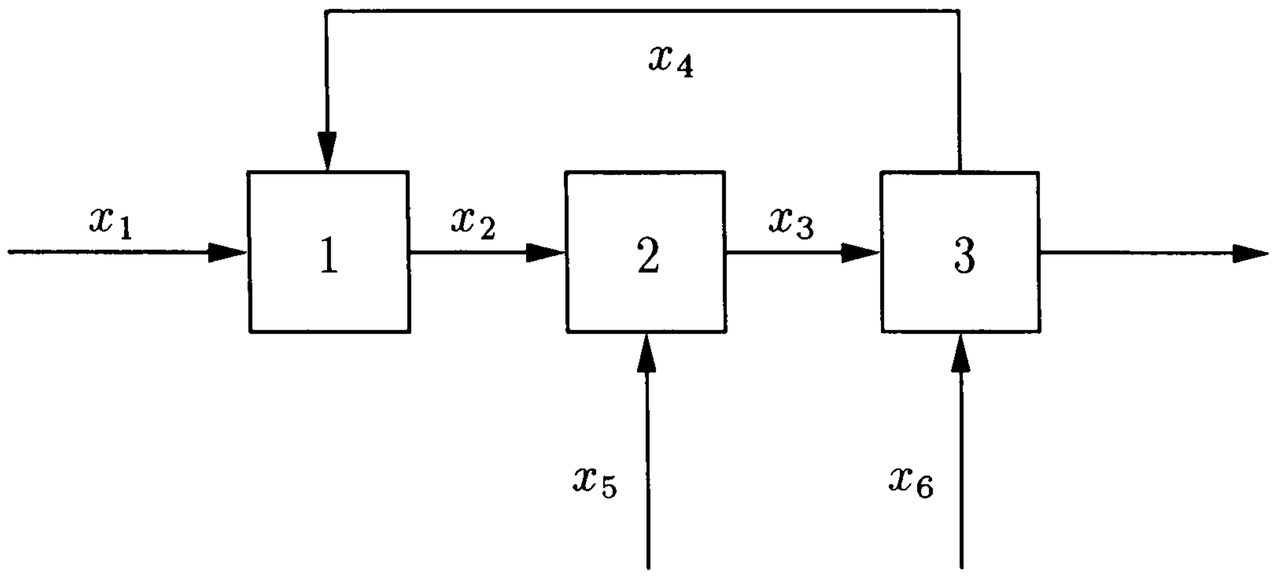


Figure C.2: Design of a Three Stage Process with Recycle

Problem **nlp3** from [57, 58]. Design of a three stage process shown in Figure C.2,

$$
\begin{aligned}
\min_{x} \quad & x_1^{0.6} + 2x_2^{0.6} - 6x_1 - 4x_3 + 3x_4 \\
\text{s.t.} \quad & \\
& x_2 - 3x_1 - 3x_3 = 0 \\
& x_1 + 2x_3 \leq 4 \\
& x_2 + 2x_4 \leq 4 \\
& x_1 \leq 3 \\
& x_4 \leq 2 \\
& x \geq 0.
\end{aligned}
\tag{C.7}
$$

The global solution, x^*, is at $[4/3, 4, 0, 0]^T$ with $f(x) = -4.514202$.

Problem **nlp4** from [57, 58][1],

$$\begin{aligned}
\min_x \quad & x_1^{0.6} + 2x_2^{0.6} + 2x_3 - 2x_2 - x_4 \\
\text{s.t.} \quad & \\
x_2 - 3x_1 - 3x_3 &= 0 \\
x_1 + 2x_3 &\leq 4 \\
x_2 + 2x_4 &\leq 4 \\
x_1 &\leq 3 \\
x_4 &\leq 2 \\
x &\geq 0.
\end{aligned} \tag{C.8}$$

The global solution, x^*, is at $[4/3, 4, 0, 0]^T$ with $f(x) = -2.2168$.

Problem **nlp6** from [57],

$$\begin{aligned}
\min_x \quad & -x_1 - x_2 \\
\text{s.t.} \quad & \\
x_2 &\leq 2x_1^4 - 8x_1^3 + 8x_1^2 + 2 \\
x_2 &\leq 4x_1^4 - 32x_1^3 + 88x_1^2 - 96x_1 + 36 \\
0 &\leq x \leq (3, 4)
\end{aligned} \tag{C.9}$$

The solution lies at $[2.3295, 3.1783]$ with an objective function value of -5.5079.

[1]Note: The problem is misprinted in [57]

Problem **fpqp3** from [57],

$$
\begin{aligned}
\min_{x,y} \quad & c^T x - \tfrac{1}{2} x^T Q x + d^T y \\
\text{s.t.} \quad & \\
2x_1 + 2x_2 + y_6 + y_7 &\leq 10 \\
2x_1 + 2x_3 + y_6 + y_8 &\leq 10 \\
2x_2 + 2x_3 + y_7 + y_8 &\leq 10 \\
-8x_1 + y_6 &\leq 0 \\
-8x_2 + y_7 &\leq 0 \\
-8x_3 + y_8 &\leq 0 \\
-2x_4 - y_1 + y_6 &\leq 0 \\
-2y_2 - y_3 + y_7 &\leq 0 \\
-2y_4 - y_5 + y_8 &\leq 0 \\
0 \leq x &\leq 1 \\
y &\geq 0 \\
y_1, y_2, y_3, y_4, y_5, y_9 &\leq 1
\end{aligned}
\tag{C.10}
$$

where $c = (5,5,5,5)$, $Q = 10\mathbf{I}$ and $d = -\mathbf{I}$. The solution is; $x^* = [1,1,1,1]^T$, $y^* = [1,1,1,1,1,3,3,3,1]^T$ with $f = -15$.

Problem **fpqp5** from [57],

$$
\begin{aligned}
\min_{x,y} \quad & c^T x - \tfrac{1}{2} x^T Q x + d^T y \\
\text{s.t.} \quad & \\
AX &\leq b \\
X &= (x,y)^T \\
0 \leq X &\leq 1
\end{aligned}
\tag{C.11}
$$

where

$$A = \begin{pmatrix} -2 & -6 & -1 & 0 & -3 & -3 & -2 & -6 & -2 & -2 \\ 6 & -5 & 8 & -3 & 0 & 1 & 3 & 8 & 9 & -3 \\ -5 & 6 & 5 & 3 & 8 & -8 & 9 & 2 & 0 & -9 \\ 9 & 5 & 0 & -9 & 1 & -8 & 3 & -9 & -9 & -3 \\ -8 & 7 & -4 & -5 & -9 & 1 & -7 & -1 & 3 & -2 \\ -7 & -5 & -2 & 0 & -6 & -6 & -7 & -6 & 7 & 7 \\ 1 & -3 & -3 & -4 & -1 & 0 & -4 & 1 & 6 & 0 \\ 1 & -2 & 6 & 9 & 0 & -7 & 9 & -9 & -6 & 4 \\ -4 & 6 & 7 & 2 & 2 & 0 & 6 & 6 & -7 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

$$b = (-4, 22, -6, -23, -12, -3, 1, 12, 15, 9, -1)^T$$

$$Q = 10\mathbf{I}$$

$$d = (10, 10, 10)$$

$$c = (-20, -80, -20, -50, -60, -90, 0)$$

The global solution is $x^* = [1, 0.907, 0, 1, 0.715, 1, 0]^T$, $y^* = [0.917, 1, 1]^T$. The objective function value is -268.015. Note that this value is incorrectly printed in the original reference [57].

Problem **fpqc2** from [57],

$$
\begin{aligned}
\min_{x} \quad & 37.293239x_1 + 0.8356891x_1x_5 + 5.3578547x_3^2 \\
\text{s.t.} \quad & \\
-0.0022053x_3x_5 + 0.0056858x_2x_5 + 0.0006262x_1x_4 &\leq b_1 \\
0.0022053x_3x_5 - 0.0056858x_2x_5 - 0.0006262x_1x_4 &\leq b_2 \\
0.0071317x_3x_5 + 0.0021813x_3^2 + 0.0029955x_1x_2 &\leq b_3 \qquad \text{(C.12)} \\
-0.0071317x_3x_5 - 0.0021813x_3^2 - 0.0029955x_1x_2 &\leq b_4 \\
0.0047026x_3x_5 + 0.0019085x_3x_4 + 0.0012547x_1x_3 &\leq b_5 \\
-0.0047026x_3x_5 - 0.0019085x_3x_4 - 0.0012547x_1x_3 &\leq b_6 \\
(78, 33, 27, 27, 27) \leq x \leq & (102, 45, 45, 45, 45)
\end{aligned}
$$

where $b = [6.665593, 85.334407, 29.48751, -9.48751, 15.699039, -10.699039]$. $x^* = [78, 33, 29.9953, 45, 36.7758]^T$ and $f^* = 1.0127 \times 10^4$.

Appendix D

Nomenclature

$\mathbb{R}$	The set of real numbers.
$\mathbb{I}$	The set of compact intervals.
n	Dimensionality. cf $\mathbb{R}^n$.
i	Components. $i = 1 \ldots n$.
N	A discrete number of points.
k	Sampled points index, $k = 1 \ldots N$, or iteration number
x	The optimisation variables. $(x \in \mathbb{R}^n)$
x_i	A component of x. $(x_i \in \mathbb{R})$
x^*	A global minimiser of $f(x)$. The solution to NCP.
$\mathcal{A}$	The feasible region. $(\mathcal{A} \subseteq \mathbb{R}^n)$
y^*	The global minimum, $f(x^*)$. $(y^* \in \mathbb{R})$
$\overline{y^*}$	Upper bound on the global minimum, y^*.
L	The Lipschitz constant. See (A.6). $(L \in \mathbb{R})$

ϵ A termination constant or error in solution. See (1.3).

$Q(x^*)$ Region of attraction of the global optimum.

a, b Scalars.

c, d Scalars.

$[\underline{x}, \overline{x}]$ A compact interval $\{x | \underline{x} \leq x \leq \overline{x}\}$.

X An interval in x. $(X \in \mathbb{I}^n)$

$w(X)$ $w : \mathbb{I}^n \to \mathbb{R}$ Width of X.

$f(x)$ $f : \mathbb{R}^n \to \mathbb{R}$. Usually the objective function.

$h(x)$ Equality constraints

$g(x)$ Inequality constraints

$F(X)$ An inclusion function for $f(x)$ on X.

$H(X), G(X)$ Inclusions of $h(x)$ and $g(x)$ respectively.

$\mathcal{V}, \mathcal{W}$ Partitions of the feasible region corresponding to v and w in GOP.

$\nabla^2 f(x)$ Hessian matrix

$\mathcal{X}^k$ A partition of the feasible region.

$J^k(x)$ Underestimator function from iteration k.

α, β, γ Positive constants. $(0 \leq \gamma \leq 1)$

$\mathcal{S}$ A compact convex set

Acknowledgments

I would like to express my gratitude to my supervisor, my department and the EPSRC for their support. I would also like to thank all the people who have helped or encouraged me with this thesis especially those who spent time reading or discussing it with me.

You know who you are,

Robert P Byrne